



# MASTER'S FINAL PROJECT



ESCOLA  
POLITÈCNICA SUPERIOR  
UNIVERSITAT DE LLEIDA  
INSPIRING THE FUTURE

**Student:** David Sánchez Marín

**Study:** Master in Computers Engineering

**Title:** Automation of the analysis of comments  
from the UdL surveys

**Director:** Roberto García González

Presentation

Month: July

Year: 2021

# Acknowledgements

I would like to thank my family for their support and patience during the development of this project and the master in general.

I also wish to appreciate the head of the Quality Office and the responsible for the surveys process for his expertise and access to the information.

I should express my gratitude to the previous survey's leader for his support and advice in the definition and validation of the application and in the development of the project in general.

And finally, I would recognize the director of the project for his guide and useful information in all the steps of the creation of the project.

# Index

Acknowledgements	1
Index	2
Index of tables	5
Index of schemas	6
Index of images	7
Introduction	9
Motivation	9
Proposal	9
Objectives	10
Structure of the document	10
State of Art	12
What is Natural Language Processing	12
Potential uses of Natural Language Processing	12
Features of a NLP system	13
Tokenization	13
Sentence detection	14
Part-of-speech	14
Dependency parsing	15
Lemmatization	15
Text classification and training model	15
Ruled-based matching	16
Tools of NLP	16
Natural Language Toolkit (NLTK)	16
TextBlob	16
Stanford Core NLP	17
AllenNLP	17
Spacy	17
Development	18
Temporary planning	18
Planning progress	19
Budget	20

Iteration 1. Definition of the user interface	21
Requirements / Specification	21
Design	21
Quantification	24
Original database	25
Implementation	27
Database	27
Agile Behaviour Driven Development (BDD)	28
Features list	28
Feature: Authenticate	29
Feature: Lista campaigns	29
Feature: List survey comments	29
Feature: View survey comment	30
Feature: Edit Comment	31
Feature: Import surveys	31
Feature: Process comments	32
Results / Conclusions	33
Iteration 2. Sample file load and language detection	34
Requirements / Specification	34
Design	36
Implementation	37
Results / Conclusions	38
Iteration 3. Analysis and improvement of language detection	40
Requirements / Specification	40
Design	40
Implementation	40
Results / Conclusions	40
Iteration 4. Model 1. Issue ‘No ha impartit classe’	43
Requirements / Specification	43
Design	43
Implementation	44
Train_model	45
Evaluate	46
Results / Conclusions	48
Iteration 5. Model 1 alternative: Parts-of-speech and Dependency parsing	50
Requirements / Specification	50

Design	51
Implementation	51
Model_1_no_classes_part-of-speech_train.ipynb	52
Model_1_no_classes_part-of-speech_evaluate.ipynb	54
Results / Conclusions	54
Iteration 6. User interface implementation	56
Requirements / Specification	56
Design	56
Implementation	56
Authentication	57
Feature: List Campaigns	58
Feature List Survey comments	60
Feature Edit Comment	62
Results / Conclusions	65
Iteration 7. Import surveys	68
Requirements / Specification	68
Design	69
Implementation	71
Multiple databases management	71
Legacy databases access	73
Campaign types and Campaigns importation	76
Import surveys	77
Results / Conclusions	81
Iteration 8. Integration of Django and Spacy - Process comments	82
Requirements / Specification	82
Design	82
Implementation	84
Creation of classes TfmLangDetect and TfmCategorizerModel1	84
Calculating language in ImportCampaign view	86
Modification of <i>Comment</i> model	87
Modification in template comments_list.html to identify language	87
Modification in templates to process issue type of comments	87
Creation of ProcessComments view	88
Definition of url to process comments	89
Summary of languages and issue types	89
Filter by language and issue type	90

Results / Conclusions	92
Iteration 9. Model 2. Problematic comments issue	94
Requirements / Specification	94
Design	94
Implementation	96
Results / Conclusions	98
Iteration 10. Integration of Django and Spacy in Model 2 - Problematic comments	102
Requirements / Specification	102
Design	102
Implementation	103
Creation of class TfmCategorizerModel2	103
Creation of Celery task process_models	103
Given information to the user	106
Changes in summary of issue types	108
Results / Conclusions	109
Conclusions	112
Future development	115
Bibliography	116

## Index of tables

Table 1 Distribution of comments by language and year	38
Table 2 Distribution of comments by question type	38
Table 3 Distribution of comments by issue type	39
Table 4 Comparative between Spacy and PyCld2	41
Table 5 Crosstable of languages in Spacy and PyCld2	42
Table 6 Distribution of comments without language	42
Table 7 Activity graph of Train model 1	43
Table 8 Distribution of comments in Training and Test datasets	45
Table 9 Distribution of data set by language and type	48
Table 10 Performance by language and model	48
Table 11 Example of Parts-of-speech	50

## Index of schemas

Schema 1 Example of tokenization task	14
Schema 2 Example of dependency parsing	15
Schema 3 Training model process	16
Schema 4 GANT graph	19
Schema 5 Structure of the Lime database	26
Schema 6 Structure of the TfmSurveys database	27
Schema 7 Activity graph of load_data	36
Schema 8 Table of confusion	46
Schema 9 Calculation of Precision (P) indicator	47
Schema 10 Calculation of Recall (R) indicator	47
Schema 11 Calculation F-score (F) indicator	48
Schema 12 Dependency parsing	50
Schema 13 Activity graph model 1 - parts-of-speech	51
Schema 14 Activity graph part-of-speech training	52
Schema 15 Dependency parsing in training model	53
Schema 16 Activity graph part-of-speech evaluate	54
Schema 17 Activity graphs user interface	56
Schema 18 Partial Lime database	69
Schema 19 Activity graph import campaigns	70
Schema 20 Activity graph import surveys and comments	70
Schema 21 Uxxienc_resul database models	74
Schema 22 Examples of Uxxienc_resul tables	74
Schema 23 Spacy classes graph	82
Schema 24 Activity graph import surveys with language detection	83
Schema 25 Model Comments modification	83

Schema 26 Activity graph Process model 1	84
Schema 27 Activity graph Model 2 processing	95
Schema 28 Comment prediction – indicators graph	98
Schema 29 Sentence prediction – indicators graph	98
Schema 30 Comments avaluation - sets graph	98
Schema 31 Sentence avaluation - sets graph	98
Schema 32 Pretrained models comparation graph	99
Schema 33 Indicators graph by comments collection	100
Schema 34 Sets size by comments collection	100
Schema 35 Indicators evolution by solution type	100
Schema 36 Indicators graph in Spanish language	101
Schema 37 Set sizes in Spanish language	101
Schema 38 Activity graph Model 2 processing	102

## Index of images

Image 1 Login screen	21
Image 2 List of campaigns screen	22
Image 3 List of surveys and comments screen	22
Image 4 Comment details screen	23
Image 5 Example of folder structure	34
Image 6 Example of sampla data	35
Image 7 Example of csv structure	35
Image 8 Example of resultant files	36
Image 9 Login screen	65
Image 10 List of campaigns screen	65
Image 11 Lista of surveys and comments screen	66
Image 12 Edit comment screen	66
Image 13 Lista of campaigns screen	68



Image 14 List of campaigns screen modification	92
Image 15 Lista of surveys and comments screen modification	92
Image 16 Edit comment screen modification	93
Image 17 Information about processing start	109
Image 18 Information about processing model 1	110
Image 19 Information about processing model 2	110
Image 20 Information process finished	111

# Introduction

## Motivation

For the completion of the Master's degree, it's necessary to carry out a Master's Final Project in which the theoretical and practical contents acquired during the master's degree are developed.

In this sense, one of my responsibilities is to support Satisfaction Surveys, which depend on the Quality and Teaching Planning unit. One of the major management issues of this unit is the review and analysis of free comments in surveys, and this would be the overall goal proposed for this Master's Final Project.

The University of Lleida conducts a wide range of types of surveys, aimed at students, faculty, administrative staff, interns in companies, ... As a general rule, all surveys include a set of closed-ended questions, where questions should be rated from 1 to 5. But they also include in almost all cases free text questions for the respondent to express their comments on the teacher, the subject, the degree, ...

These comments are reviewed and analyzed periodically at different levels within the university, such as from the Quality unit, the degree coordinators, the people in charge of the center or the vice-rectorate responsible for teaching.

The problem is that a percentage of responses include foul-sounding and offensive expressions that are not considered appropriate to use and disseminate. To solve this problem the staff of the Quality unit performs a manual revision of the comments eliminating derogatory sentences or replacing them with expressions with the same meaning as expressed more correctly.

## Proposal

The aim I propose for this Master's Final Project is to develop a tool to automate the process of consultation, review, publication and analysis of survey comments. It will be oriented to natural language processing and also to "sentiment analysis".

This tool should provide the following feature blocks:

- Consultation of the initial status of the comments and assessment of the surveys to be treated.
- Artificial Intelligence technology treatment of selected comments with the aim of eliminating completely inadmissible comments, proposing formally correct alternative comments and assessing or classifying the meaning of the sentence.
- Visualization of the result of the revision with the possibility of manually retouching the final sentences.
- Detection of the most used expressions (trending topics).
- Numerical and graphical analysis of the meaning of the results at different levels with the possibility of modifying the queries by the user.

## Objectives

Other objectives of this work are:

- Transversal use of the technologies presented throughout the Master.
- Integration, if possible, with the technologies used by the University's Information Systems
- Regarding data privacy with tools for naming and drafting the necessary legal agreements with the data controller
- Exploring the state of the art in Artificial Intelligence with regard to text analysis
- Monitoring of standards for the development of Business Intelligence applications and projects

Other requirements:

- Language: Contents of comments could be in Catalan, Spanish or English. The language must be detected dynamically, as the source application does not provide this information.
- Database: The database will be MySQL for compatibility with the data source system.
- The user interface will be in Catalan.

## Structure of the document

In this document, I will describe the development of the project of review and analysis of comments from the surveys of the UdL.

The tool developed will use Natural Language Processing (NLP) techniques and programming of web applications.

The first part of the document will expose the state of the art in Natural Language Processing, the existing tools to develop applications that use this technology, and the characteristics of the library selected to create the application.

Will also be explained the planification of the project and its evolution.

The most significant part of the document is describing the successive iteration done in implementing the project. These iterations are:

- Iteration 1: Definition of the user interface

Firstly, the document describes the original information system where surveys are managed, called Lyme Surveys. This description includes the structure of its database. It will also be quantified the volume of information involved in managing the surveys and his comments. As well will be designed the structure of the database of the project. And finally, it will be defined the features and scenarios of the project, following the Behaviour Driven Development (BDD) methodology.

- Iteration 2: Sample file load and language detection

Secondly, it's told how the information from the original databases has been loaded, transformed, and organized to be used in the NLP processing. Also a little analysis is done about the composition of this information.

- Iteration 3. Analysis and improvement of language detection

Next, it's exposed how has been solved the problem of detecting the language of the comments. This detection will be done with a combination of two libraries: LanguageDetect of Spacy and PyCld2. It will be analyzed the limitations of both libraries and proposed a mix that improves the results.

- Iteration 4. Model 1. Issue 'No ha impartit classe'

Then, it will be presented the implementation of the first Spacy NLP model, using the class TextCategorizar. This model must detect comments that could be classified in the issue type: 'Professor hasn't taught this group'. We will use comments labeled by the user to train a model for each language and save it from being used by the web application. It will also be analyzed the performance of the model.

- Iteración 5. Model 1 alternative: Parts-of-speech

In this iteration, it will be exposed a second method of detecting the comments included in the previous issue type, using Part-of-speech and Dependency Parsing. With this method, I try to increase detection efficiency when having little original information to train the model.

- Iteration 6. User interface implementation

After, it will describe the implementation of the user interface using the framework Django. This implementation includes all the basic features, and it works with a set of test data. It doesn't include the importation of new information from the source application or the processing of the comments.

- Iteration 7. Import surveys

This block will explain the feature of importing surveys and comments from the source application Lime. It's described how some challenges are solved such as managing multiple databases, accessing legacy databases, using multiple key tables, and some other subjects.

- Iteration 8. Integration of Django and Spacy - Process comments

This part will expose the integration of the NLP Spacy model created in iteration 4 in the user interface developed with Django. It will be shown the creation of some classes based on the results of the Spacy Notebooks and the modification of the user interface to start the comments processing and show the results.

- Iteration 9. Model 2. 'Problematic comments' issue

This section will explain the creation of a new Spacy NLP model that will classify comments corresponding to issue type 2, 'Problematic comments'. It will be tried some possible implementations and compared between them, testing his performance.

- Iteration 10. Integration of Django and Spacy in Model 2 - Problematic comments

This is the last iteration and consists of integrating the NLP Spacy models created in the previous iteration inside the user interfaces. The specific part of this iteration is the use of Celery tasks and the feedback to the user.

And finally, will be exposed the conclusions of the project and the proposal for future developments.

## State of Art

In this section will be described the related technologies, studies performed to complete the work and other similar solutions, etc.

### What is Natural Language Processing

Natural Processing Language (NLP) is a subfield of artificial intelligence which studies how computers process and analyze large amounts of natural language data. The aim is to create tools that allow computers to understand the document contents and its contextual information. NLP combines rule-based modeling from linguistic science with statistical, machine learning, and deep learning models to automatically extract, classify and label elements of text and voice.

### Potential uses of Natural Language Processing

Some of the tasks of an NLP system are:

**Speech recognition.** This is the task of converting voice into text data to execute voice commands or answer spoken questions. It may lead to the speed of the voice, intonation, incorrect grammar, etc.

**Part of speech tagging.** This function is to identify the different parts of a speech and classify it based on its use and context. It can indicate if a word is a noun or a verb if it is a complement of another word, ...

**Name entity recognition.** This function identifies if a word corresponds to an object in the real world and what type of object. For example, a person, a city, a company, ...

**Word sense disambiguation.** It consists of selecting the meaning of a word between multiple meanings using semantic analysis and its context.

**Sentiment analysis.** It tries to obtain the subjective valuation of a sentence and the mood of the emitter.

**Natural language generation.** This is the translation of computer information into human language.

**Automatic summarization.** It consists of detecting the main words or ideas of a text, creating a new text including it, and expressing it in human language.

## Features of a NLP system

An NLP library or platform could implement some of the below features:

NAME	DESCRIPTION
Tokenization	Segmenting text into words, punctuation marks etc.
Part-of-speech (POS) Tagging	Assigning word types to tokens, like verbs or nouns.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Lemmatization	Assigning the base forms of words. For example, the lemma of “was” is “be”, and the lemma of “rats” is “rat”.
Sentence Boundary Detection (SBD)	Finding and segmenting individual sentences.
Named Entity Recognition (NER)	Labelling named “real-world” objects, like persons, companies or locations.
Entity Linking (EL)	Disambiguating textual entities to unique identifiers in a Knowledge Base.
Similarity	Comparing words, text spans and documents and how similar they are to each other.
Text Classification	Assigning categories or labels to a whole document, or parts of a document.
Rule-based Matching	Finding sequences of tokens based on their texts and linguistic annotations, similar to regular expressions.
Training	Updating and improving a statistical model’s predictions.
Serialization	Saving objects to files or byte strings.

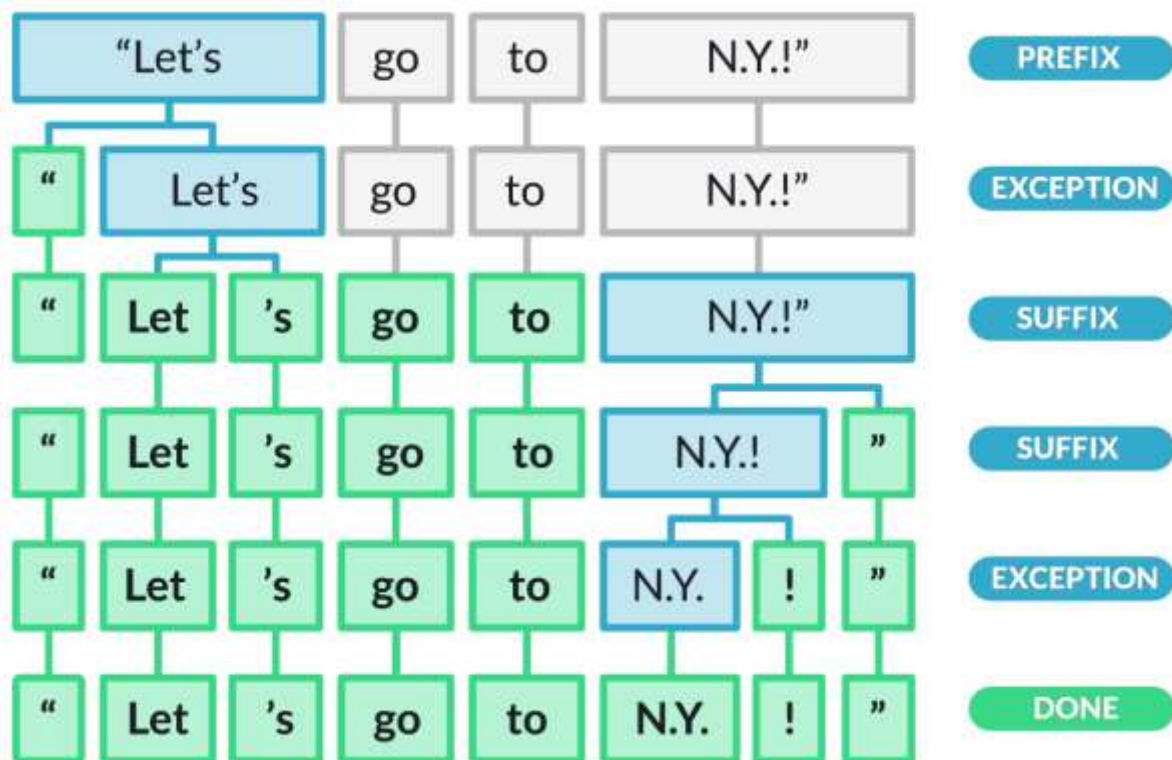
The following sections will detail the features needed in this project.

### Tokenization

Tokenization is the action of segmenting the text into words, punctuations, stopwords and other symbols. The basic technique is to split text on whitespace characters. But also can be more sophisticated and use the syntactic rules of every language to divide the text into smaller parts or group words in one unit.

Stopwords are small words in a language that doesn't provide additional meaning to the sentence and are usually removed in an initial analysis. For example, articles, conjunctions, connectors, etc.

This is the first and more basic step in the analysis of a text.



Schema 1 Example of tokenization task

### Sentence detection

This task is a type of tokenization consisting in detecting the start and end of sentences in the text. It could be done using the punctuation characters or using a more complex approach using the syntax and grammar to detect the structure of the sentence.

### Part-of-speech

After splitting text into words, the system must identify the parts of speech. This action means deciding grammatically what type of word is. For example, noun, verb, article, etc

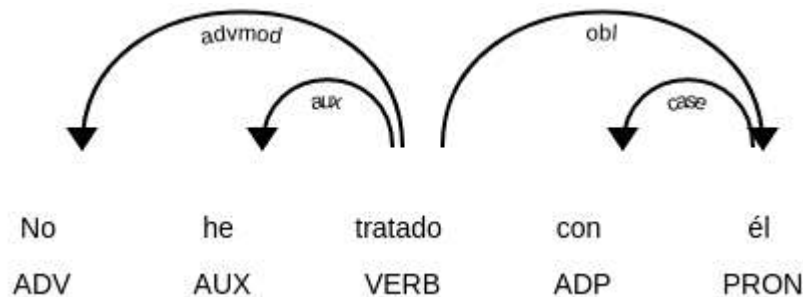
This step can also include deciding the function of the word in the sentence. For example, subject, complement, predicate, etc.

And finally, result of this process could also be called Linguistic annotations.

To do this task, the tool must have a statistical model that makes the prediction of the type, function and dependencies of the word. This model has to be trained with an extensive corpus of sentences and is the system's core.

## Dependency parsing

Some systems also detect the dependencies or relations between words. For example, what noun complements an adjective.



Schema 2 Example of dependency parsing

## Lemmatization

Lemmatization is the process of extracting from the word its base form. This process removes suffixes and prefixes, but also considers the context and meaning of the word, to select the correct form of the word.

For example, the lemma of 'working' is 'work' and the lemma of 'are' is 'be'.

## Text classification and training model

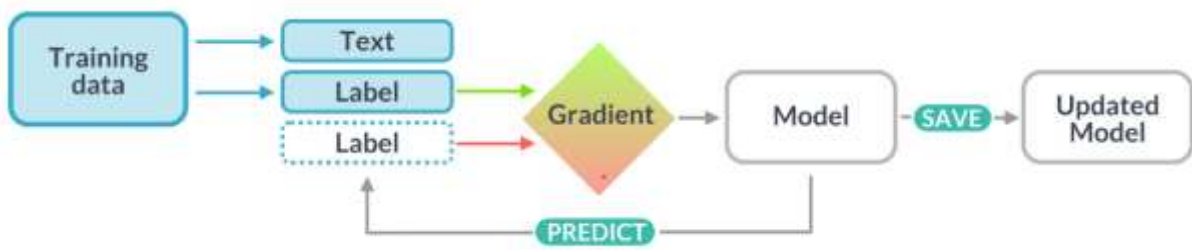
Text classification is the process of categorizing text into different groups using a set of sample text labeled by a user. Every text of this sample set has a label indicating if it's included in the classification or not.

The NLP tool will create a statistical model that will be trained using the sample information. This statistical model could be a new model, or it could be based on a pretrained model. This is useful when the training set is not big enough.

The model estimates weight values based on examples during training. Training is an iterative process in which the predictions are compared with the labels of the sample data, and this information is used to calculate the gradient of the loss. And this gradient is then used to adjust the weights and indicates how the weight values should be changed, so the predictions become more similar to the sample labels.

After the training, the model must be evaluated using a new collection of user-labeled texts to calculate the accuracy of the classification.





Schema 3 Training model process

### **Ruled-based matching**

Rule-matching is the process of defining patterns based on the contents and grammar of the sentence to identify the text that could match with defined patterns.

A pattern is similar to the regular expression of Python and other languages but using the grammar of words. For example, a sentence that starts with a noun and an adjective and contains a verb conditional. Or the lemma of the verb is 'run' to include all the tenses of the verb.

Patterns usually are defined analyzing the part-of-speech of the sample texts.

## **Tools of NLP**

Below we will see some of the main NLP tools.

### **Natural Language Toolkit (NLTK)**

NLTK is a platform for building Python programs that works with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet. Also includes a suite of text processing libraries for tokenization, classification, tagging, semantic reasoning, etc. NLTK is suitable for linguists, engineers, students, researchers, and industry users. It comes from the University of Pennsylvania, and it's initially oriented to learning.

### **TextBlob**

TextBlob is a Python library for processing textual data. It provides a API to execute the Natural Language Processing (NLP) tasks such as tokenization, part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, etc.

It is built on the shoulders of NLTK library. It's easy to start in it for beginner users due to its simple interface. It uses syntax similar to manipulating Python strings.

## **Stanford Core NLP**

Stanford CoreNLP is an integrated framework, which makes it very easy to apply a bunch of language analysis tools to a piece of text. The Stanford CoreNLP code is written in Java and licensed under the GNU General Public License

CoreNLP enables users to derive linguistic annotations for text, including tokenization and sentence boundaries, parts of speech (POS), named entities (NER), numeric and time values, dependency and constituency parses, coreference, sentiment, quote attributions, and relations. CoreNLP currently supports 6 languages: Arabic, Chinese, English, French, German, and Spanish.

The tools variously use rule-based, probabilistic machine learning, and deep learning components.

## **AllenNLP**

AllenNLP is a natural language processing platform for building state-of-the-art models. It's a free, open-source project from AI2, built on PyTorch. It makes it easy to design and evaluate deep learning models. It provides the infrastructure for run them in the cloud or on a laptop. It's built and maintained by the Allen Institute, the University of Washington and open-source development community. AllenNLP is designed to support researchers, engineers, students, etc

## **Spacy**

SpaCy is a free open-source library for Natural Language Processing in Python. It is designed specifically for production use and helps build applications that process large volumes of text. It can be used to build information extraction or natural language understanding systems, or to pre-process text for deep learning. It includes features as tokenization, part-of-speech tagging (POS), dependency parsing, lemmatization, sentence boundary detection, name entity recognition (NER), similarity, text classification, rule-based matching, training and serialization.

# Development

## Methodology

This project will be developed in successive iterations where it will be implemented a group of functionalities more or less independents.

The source code and documentation will be stored in a repository of GitHub.

The address of the repository is:

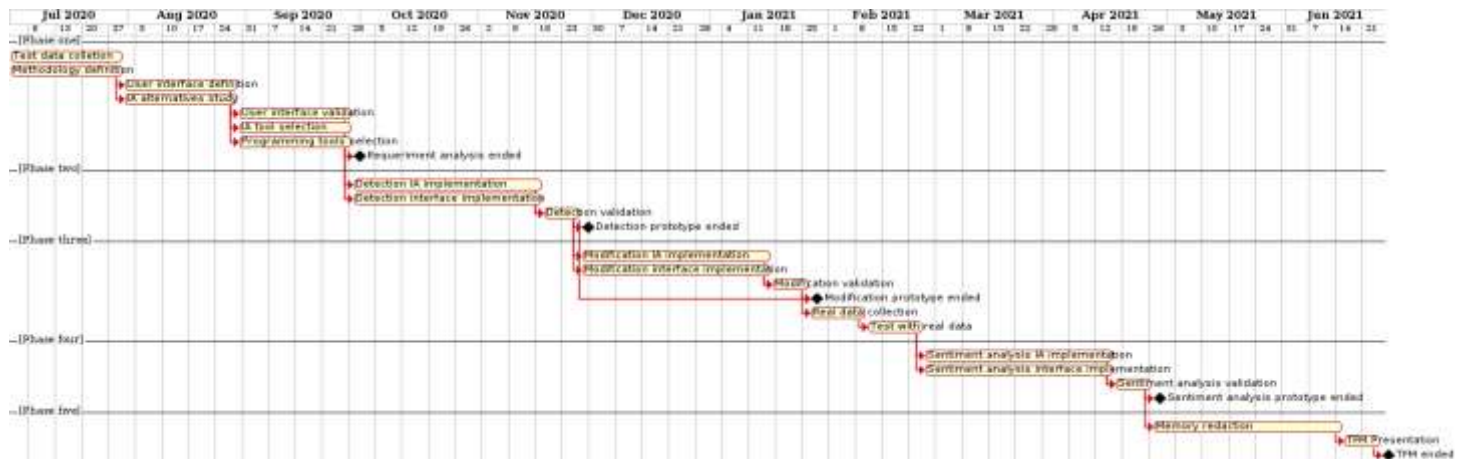
***<https://github.com/dsm9/TreballFiMaster>***

## Temporary planning

The initial planning proposal would be:

- **July 2020:** Collection of original data to define the model and perform the tests. Definition of working methodology and coordination with the actors involved.
- **July - August 2020** - Initial user interface definition and functional application requirements. Study of existing alternatives, tools and methods in the market related to Natural Language Processing and Sentiment Analysis.
- **September 2020:** Validation of user functionalities and interface. Selection of the main tools to be used in the analysis of the texts. Selection of interface programming tools.
- **October - November 2020:** Validation of user functionalities and interface. Implementation of the interface part for displaying conflicting comments and classification.
- **December 2020 - January 2021:** Development of the module for proposal of modification or removal of sentences. Implementation of the interface part for displaying and validating proposals.
- **February 2021:** Collection of new information for test. Test of previous modules with real information and test with real users.
- **March - April 2021:** Development of the sentiment analysis module. Implementation of the interface for consultation of the sentiment analysis.
- **May - June 2021:** Drafting of the project memory and other documentation for your Presentation. Last adjustments in the IA programming and the interface according to user proposals.
- **June 2021:** Defense of the Master's Final Work. Implementation in production.

This is the GANT graph of the proposed planification:



Schema 4 GANT graph

## Planning progress

- July 2020:** Collection of original data to define the model and perform the tests.  
 Definition of working methodology and coordination with the actors involved.  
 SpaCy research, to decide if it can be useful for analysis. The Spacy development would be implemented using the Jupyter Notebook tool. Search for an extension for the Catalan language.
- August 2020:** Initial definition of the user interface and functional requirements of the application, adapting the prototype to the proposals of the users.  
 Deepen the tests with spaCy, search for an extension for automatic language detection.  
 Quantification of the volume of data (surveys, responses, comments, ...) Preparation of data sets for model training.
- Setembre 2020:** Validation of the interface between users.  
 Selection of analysis and design (UML) and preparation of basic diagrams.  
 Selection of the programming tool for the user interface. It will be the language Python and the Django programming environment.  
 Definition of the database structure that will be implemented with MySQL.
- October 2020:** Start of the users' interfaces development.  
 Database implementation.
- November 2020:** Continue with the evolution of the user interface. Creation of all the screens and the flow between them. It's a prototype. The processes of the information in the transition between screens remains pending. Corresponding to iteration 1.
- December 2020:** Deepen in the knowledge about Spacy and search for examples of

similar use cases.

Revision of the original data files to adapt them to the concrete format that will need the NLP analysis.

Creation of the Jupiter Notebooks used to load the initial data and to detect the initial language of the comments, corresponding to iteration 2.

- **January 2021:** Analysis and enhancement of the language detection, corresponding to iteration 3.  
First, implementation of a Natural Language Processing (NLP) model with the capacity to detect comments that match with issue type 1 - Professor haven't done classes, using *TextCategorizer* and corresponding to iteration 4.
- **February 2021:** Second implementation of the NLP model to classify the issue type 1, in this occasion using *Part-of-speech* and *Rule-matching*, comparison of results with option *TextCategorizer* and selection of the first option as the final selection. This work corresponds to iteration 5.
- **March 2021:** Enhancement of the user interface to work with information from a test database and complete interaction between screens, resulting in an operative test application for the user. Corresponds to iteration 6.  
Implementation of the data importation from the legacy databases (Lime), work with multiple databases, virtual models and adaptation of the user interface to activate these functionalities. Corresponds to iteration 7.
- **Abril 2021:** Integration of Django and Spacy to classify the comments of the issue type 1 - Professor hasn't done classes. Creation of classes that use Spacy functions and modification in the user interface to execute the classification and show results. Corresponds to iteration 8.
- **May 2021:** Implementation of a Spacy NLP model to classify the comments of issue type 2 - Problematic comments. Analysis of alternatives and enhancement of results. Corresponds to iteration 9.

## Budget

It is not possible to make an economic evaluation of the project, because the development is done voluntarily with an academic objective.

The global value in working hours, estimating a dedication of 10 hours per week is:

10 hours / week x 4 weeks x 12 months = 480 hours

## Iteration 1. Definition of the user interface

### Requirements / Specification

Within this iteration, we will proceed to describe the current computer system and identify the data sources of this project, as well as the manual comment review procedure used so far.

Based on this information, an Artificial Intelligence module will be developed that allows the detection and classification of conflicting comments. After reviewing the results of the current manual review in the last 3 courses the type of comments that require intervention are:

- The teacher who has not taught this group
- Teacher comment on subject question
- Subject comment in teacher's question
- Spelling mistakes
- Exclamation points and excessive emoticons
- Problematic comments (do not require intervention, but highlight them)
- Offensive comments. Possible actions:
  - Complete removal
  - Partial removal
  - Substitution

The document '*assignatura\_professor\_comentaris\_exemples.xlsx*' includes some samples of the different types of issues.

It will also develop the necessary web application to consult the original comments and show the classification resulting from the previous process.

### Design


Below is the prototype of some screens of the comment editing application:



The image shows a login screen prototype. At the top, there is a dark blue header bar. On the left side of the header is the Universitat de Lleida logo, which consists of a stylized 'U' with a cross above it. To the right of the logo, the text 'Universitat de Lleida' is written in white. Further to the right, the text 'Enquesta virtual' and 'Anàlisi i revisió de comentaris' is displayed in white. Below the header, the main content area is white. In the center of this area is a dark blue rectangular box containing a login form. The form has two input fields: the first is labeled 'Usuari:' and the second is labeled 'Contrasenya:'. Below these fields is a button labeled 'Entrar'. At the bottom of the screen, there is a dark blue footer bar containing the text 'Universitat de Lleida - Màster en Enginyeria Informàtica - Treball Final de Màster - David Sánchez Martí' in white.

Image 1 Login screen


First, the user will be validated using the UdL user and password.



Universitat de Lleida

Enquesta virtual

Anàlisi i revisió de comentaris



Selecció de campanya:

Mostrar

10

registres

Buscar:

Cod. campanya	Nom campanya	tipus Campanya	Data extracció
101	18-19 Assignatura-professor màsters P 1r S	Enquesta assignatura-professor grau i màster univ	2019-10-18 12:45:32
102	18-19 Assignatura-professor màsters NP 1r S	Enquesta assignatura-professor grau i màster univ	2019-04-10 13:49:19
103	18-19 Assignatura-professor INEFC 1r S	Enquesta assignatura-professor grau i màster univ	2019-04-10 14:53:01
105	18-19 Titulats Grau AQU	Enquesta de final de programa - Grau	2020-02-03 14:06:27
106	18-19 Titulats Doble Grau AQU	Enquesta de final de programa - Grau	2020-02-03 14:16:32
108	18-19 Doctorat - Estudiantat doctor	Enquesta de doctorat - Estudiantat doctor	2020-02-03 14:22:22
109	18-19 Pràctiques externes - estudiantat (GEM - 1P)	Enquesta de Pràcticum - Estudiantat	2019-03-07 12:26:10
110	18-19 Titulats de Màster AQU	Enquesta de final de programa - Màster AQU	2020-02-03 14:20:13
111	18-19 Titulats formació contínua 1r P	Enquesta de formació contínua	2020-01-28 13:55:27
112	18-19 Assignatura-professor màsters P 2n S	Enquesta assignatura-professor grau i màster univ	2019-09-25 10:46:34

Mostrando registros del 1 al 10 de un total de 64 registros

Anterior

1

2

3

4

5

6

7

Siguiente

Universitat de Lleida - Màster en Enginyeria Informàtica - Treball Final de Màster - David Sánchez Marín

Image 2 List of campaigns screen

Then the user can choose the survey campaign to deal with among all the exported ones.

Universitat de Lleida

Enquesta virtual

Anàlisi i revisió de comentaris

10

0

Campanya:

18-19 Assignatura-professor màsters P 1r S (101)

Tipus campanya:

Enquesta assignatura-professor grau i màster univ.

Data exportació:

2019-10-18 12:45:32

Desar comentaris acceptats

Resum campanya:

Enquestes: 114

Enquestats: 1.488

Respostes: 11.775

Comentaris: 640

Mostrar 10 registres

Cercar:

ID	Descripció	Codi pregunta	Pregunta	ID	Comentari original	Tipus incidència (Classificació)	Proposta solució	Comentari proposat	Acceptar
1020	2018-19 - 1 - M04 - MJ en Recerca en Salut - 1854 - ANÀLISI DE DADEN EN LA INVESTIGACIÓ EN SALUT 1 - 1 - Test 1 ANÀLISI DE DADEN EN LA INVESTIGACIÓ EN SALUT 1	A02	COMENTARIS DE L'ASSIGNATURA (aspectes positius / aspectes de millora)	1	Molt bon mestre. Respon molt ràpid i molt detallat.	Passee a professor	Passee a professor	(Idem)	<input checked="" type="checkbox"/>
1029	2018-19 - 1 - M01 - MJ en Sistema de Justícia Penal - 1352 - PROCEDIMENTS JUDICIALS PENALS - 1 - Test 1 PROCEDIMENTS JUDICIALS PENALS	A02	COMENTARIS DE L'ASSIGNATURA (aspectes positius / aspectes de millora)	1	Las docentes que cobren los módulos de Derecho Internacional Humanitario y Procesos Penales cobrados que tienen un excelente manejo del tema y se relacionan muy bien con los estudiantes, especialmente los dejaron muy pocas dudas para cada sus temas.	Passee a profesor	Passee a profesor	(Idem)	<input checked="" type="checkbox"/>
1032	2018-19 - 1 - M01 - MJ en Salut de Gènere i Salut de Polítiques d'igualtat - 1861 - HISTÒRIA DE LES RELACIONS DE GÈNERE - 1 - Test 1 HISTÒRIA DE LES RELACIONS DE GÈNERE	P0021	COMENTARIS SOBRE (COMENTARIS PROPOSAT) (APLLODOPROFES) (APLLODOPROFES) (aspectes positius / aspectes de millora)	1	Feedback ràpid i proper. Molt interessant.	Passee a assignatura	Passee a assignatura	(Idem)	<input checked="" type="checkbox"/>
1039	2018-19 - 1 - M01 - MJ en Sistema de Justícia Penal - 1352 - PROCEDIMENTS JUDICIALS PENALS - 1 - Test 1 PROCEDIMENTS JUDICIALS PENALS	P0022	COMENTARIS SOBRE (COMENTARIS PROPOSAT) (APLLODOPROFES) (APLLODOPROFES) (aspectes positius / aspectes de millora)	1	No he tingut temps.	No he tingut temps	Elimine registre	(Idem)	<input checked="" type="checkbox"/>
1040	2018-19 - 1 - M01 - MJ en Ensenyament d'Ensenyament de la llengua catalana - 1412 - NOVES TECNOLOGIES APLICADES A L'ENSENYAMENT DE LA LL - 2 - Test 1 Còpia NOVES TECNOLOGIES APLICADES A L'ENSENYAMENT DE LA LL	A02	COMENTARIS DE L'ASSIGNATURA (aspectes positius / aspectes de millora)	1	Aun no se ha buido. No puedo evaluar.	No he tingut temps	Elimine registre	(Idem)	<input checked="" type="checkbox"/>
1050	2018-19 - 1 - M02 - MJ en Ensenyament d'Ensenyament de la llengua catalana - 1412 - NOVES TECNOLOGIES APLICADES A L'ENSENYAMENT DE LA LL - 2 - Test 1 Còpia NOVES TECNOLOGIES APLICADES A L'ENSENYAMENT DE LA LL	A02	COMENTARIS DE L'ASSIGNATURA (aspectes positius / aspectes de millora)	1	No he tingut temps.	Faltos de tiempo	Conseguir datos	(Idem)	<input checked="" type="checkbox"/>

10

0

10

0

10

0

10

0

Image 3 List of surveys and comments screen

All the comments of the campaign, the classification assigned to it by the algorithm, and a proposed modification will be shown later.

The classification labels shall be:

- 'No ha impartit classe a aquest grup'
- 'Comentari d'assignatura'
- 'Comentari de professor'
- 'Faltes d'ortografia'
- 'Exclamacions o emoticones excessius'
- 'Comentari problemàtic'
- 'Comentari ofensiu'

Possible solutions will also be predefined according to the incidence classification.

The user must review this classification and comment proposal, being able to edit freely. This action will be done from the comment editing page.

**Universitat de Lleida** Enquesta virtual  
Anàlisi i revisió de comentaris

Campanya: 18-19 Assignatura-professor màsters P-ir S (101) Tipus campanya: Enquesta assignatura-professor grau i màster univ. Data exportació: 2019-10-18 12:45:32

id: 1028 2018-10-11 R10 MIR en Ensenyament d'Espanyol/Català per a Immigrants - 14117 ENSENYAMENT DE LA COMPETÈNCIA CULTURAL I DESENVOLUPAMENT DE LA INTERCULTURALITAT I LA MEDIACIÓ - 2 Teo2018v ENSENYAMENT DE LA COMPETÈNCIA CULTURAL I DESENVOLUPAMENT

Codi pregunta: P0202 COMENTARIS SOBRE {HOMBREPROF2.value} {APELLIDO1PROF2.value} {APELLIDO2PROF2.value}: (aspectes positius / aspectes de millora)

Comentari original:  
Es necesario realizar tantos dias de deontologia??? se han dedicado ha leer el puqeteno código!!!! para eso!!! me quedo en casa y lo leo YO!!!! que para algo se leer solta!!!!  
5 horas!!!! con una hora de media "parte!!!! PARA QUE CO#GO ?????????? hagan la classe de 4 horas y 15 minutos descanso, que no necesitas tanto tiempo para comerse el bocadillo e ir al bar!!! la gente trabaja ja a estas alturas!!! quiere ir por faena, y descansar, descansamos en NUESTRA CASA!!!!!!!!!!!!!!!!!!!!asi que esa hora ESTUPIDA que dejan entre medio..... ningun sentido.  
Despues los profesores, terminan a las 8 igualmente, y estan una hora allí hablando de su vida ..... y no de su vida practica como abogados.....

Tipus incidència:  
Comentari ofensiu  
Emoticones excessives

Proposta solució:  
Eliminar frase completa  
Eliminar Emoticones

Comentari proposat:  
Es necesario realizar tantos dias de deontologia? para eso me quedo en casa y lo leo yo que para algo se leer solta.  
5 horas con una hora de media "parte. hagan la classe de 4 horas y 15 minutos descanso, que no necesitas tanto tiempo para comerse el bocadillo e ir al bar. La gente trabaja ja a estas alturas. Quiera ir por faena, y descansar, descansamos en nuestra casa. Asi que esa hora que dejan entre medio, ningun sentido.  
Despues los profesores, terminan a las 8 igualmente, y están una hora allí hablando de su vida y no de su vida practica como abogados.

Acceptar Cancel·lar

Universitat de Lleida - Màster en Enginyeria Informàtica - Treball Final de Màster - David Sánchez Parra

Image 4 Comment details screen



## Quantification

Within this section, we will see the central values that will allow us to assess the size of the problem we face and influence the adoption of one solution or another.

The problem to be solved is the treatment of the subject-teacher surveys carried out all the courses to the degrees and masters that the university offers. All undergraduate and master students answer these surveys for each of the subjects they take. There are also other types of surveys, such as those answered by students, academic managers and business managers of external internships, those answered by graduate students or by doctoral directors and students.

The University of Lleida in the course 19-20 has taught:

46 Degrees, 13 Double Degrees and 34 Masters.

In the university the course indicated have studied:

8,234 undergraduate students and 861 master's students.

There are 2,752 trainees.

A total of 1,392 undergraduate students and 399 master's students have graduated.

There are 664 doctoral students and 33 students have read the thesis.

A total of 1,463 subjects have been taught in the degrees and 477 in the master's degrees.

Combining the information of the number of students and the subjects, the concept of the surveyed population arises, where a student counts once per subject enrolled in the course, and that is therefore the number of possible surveys that can be answered.

The population surveyed was 69,184 in the grades and 4,653 in the master's degrees, taking into account only the subject-teacher surveys.

Each survey contains a free comment question about the subject and one for each teacher in the subject, which means more than **150,000 possible comments** in an academic year.

So far, we have obtained the maximum number of possible comments to discuss, but not all surveys are answered and not all include comments.

Of these possible surveys, 29,041 responses were obtained in the grades and 2,205 in the master's degrees. This represents a participation rate of 42 per cent and 47 per cent, respectively.

In 2019-20 **14,033 comments** were received, 12,675 in the grades and 1,358 in the masters, which represents 44% and 62% of responses with comments.

Some action was taken in 343 (2.44%) of these comments. It is not a large number, but in order to detect these cases, all the comments have to be reviewed.

If the tool to be developed performs the task of detection would be an important help by focusing the effort on comments that require some intervention.

## Original database

Information from questionnaires and survey responses is contained in a MySQL database.

This database has two schemas: *lime* and *encuestas*.

The tables that allow constructing the questionnaire are:

- *lime.lime\_groups*: Groups of questions.
- *lime.lime\_questions*: Texts and properties of questions.
- *lime.lime\_answers*: Possible answers

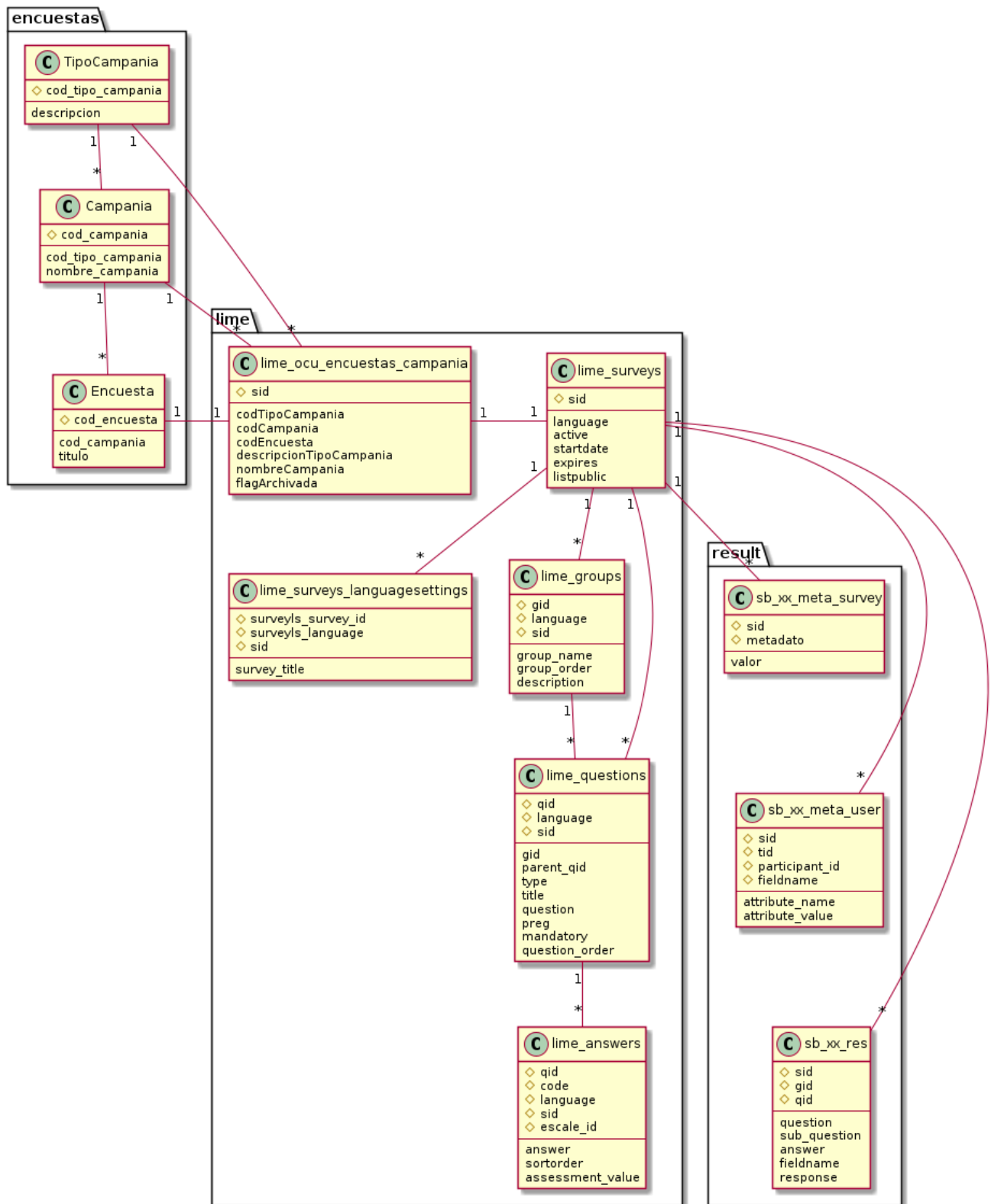
The table providing the survey list is: *lime.lime\_surveys*

The table containing the list of campaigns is: *encuestas.campania*

The table linking surveys and campaigns is: *lime.lime\_ocu\_encuestas\_campania*

The tables that are created with the survey export process are:

- *lime.sb\_?\_survey*: General survey data. Fields or metadata by which the information is grouped.
- *lime.sb\_?\_meta\_user*: Information from survey participants.
- *lime.sb\_?\_res*: Answers to each survey question by each participant.  
(? should be replaced by the campaign code).



Schema 5 Structure of the Lime database

This is a simplified scheme where tables that are not relevant for this work have been removed and only the identifying fields and those necessary for the treatment of the results are included.

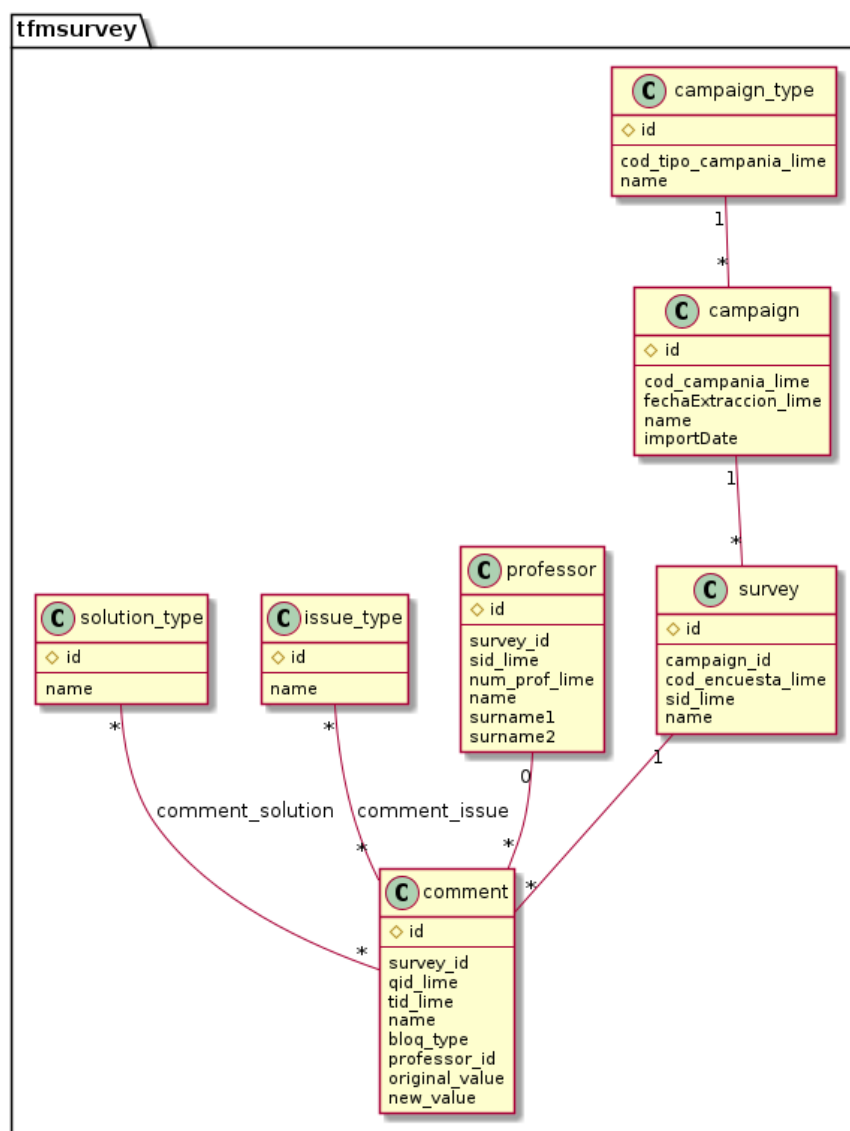
## Implementation

### Database

In order to manage the results, a new database has been created to transfer the campaign information to be processed. The result of the classification of the responses and the final values processed shall be stored in this database.

This database has been developed using MySQL, to facilitate compatibility with the current database.

The structure of this database is as follows:



Schema 6 Structure of the TfmSurveys database

## **Agile Behaviour Driven Development (BDD)**

In the development of this project, I will use a Behaviour Driven Development approach (BDD). Behaviour Driven Development is an agile software development methodology that comes from the Test Driven Development (TDD) methodology. It addresses the communication among developers, testers and domain experts and focuses on the definition of User Histories through 'Features' and 'Scenarios'.

This methodology helps us identify the features that must be implemented by our application, decide the priority on implementing these features, and facilitates acceptance testing to check if the application executes the behaviors defined.

Some of the characteristics of the Behaviour Driven Development are:

- Describe using a language accessible to domain experts, testers and developers.
- Develop the features that produce the most benefit to the result of the project.
- Describe from a top-level of abstraction to a high level of detail.
- For every user story or feature, we must define his name (Feature), the target of the feature (In order to), the stakeholder who executes it (As a), and the description (I want to)
- In the detailed level of definition, we will define the scenarios, a specific situation of use of a feature. For every scenario, we must define the pre-condition (Given), the event that initiates the scenario (When) and the outcome (Then)
- We will use the implementation of the scenarios to test the correct behavior of the feature.
- 

### **Features list**

Following the Behaviour Driven Development the features of the project have been defined:

The goal of this application it's manage the revision, analysis and modification of comments in the surveys of the UdL.

The intended features of the application are:

- Authenticate
- List campaigns
- List survey comments
- View survey comment
- Edit comment
- Import surveys
- Process comments

In the following sections, we would see the definition of the features and his main scenario.

## Feature: Authenticate

**Feature:** Authenticate user

In order to access to the application

As a user

I want to enter my user and password

**Background:** There is a registered user

**Given** Exists a user "user" with password "password"

**Scenario:** Successful login

**Given** the application has a correct connection to Lime database

**When** I enter the user "user" and the password "password"

**And** user and password are correct

**Then** I see the default page of the app

## Feature: Lista campaigns

**Feature:** List Campaigns

In order to list the campaigns imported from Lime

As a user

I want to see a list of all imported campaigns

**Background:** There is a registered user

**Given** Exists a user "user" with password "password"

**Scenario:** Select a campaign from a list

**Given** I login as user "user" with password "password"

**And** A set of campaign types had been imported from Lime

id	cod_tipo_campania_lime	name
7	7	Enquesta de final de programa - Grau

**And** A set of campaigns had been imported from Lime

id	cod_campania_lime	fecha_extraccion_lime	name	import_date
101	101	2019-10-18	18-19 Assignatura-professor màsters P 1r S	
2020-11-06	25			

**When** I list the campaigns

**Then** I'm viewing a list of campaigns

Codi campanya	Nom campanya	Codi tipus campanya	Tipus campanya
168	19-20 Pràctiques externes - estudiantat (GEM - 1P)	9	
	Enquesta de Pràcticum - Estudiantat	Nov. 17, 2020	

## Feature: List survey comments

**Feature:** List survey comments

In order to list the surveys and comments associated with a campaign

As a user

I want to see a list of all comments included in the surveys of the campaign

**Background:** There is a registered user

**Given** Exists a user "user" with password "password"

**And** I login as user "user" with password "password"

```

Scenario: Select a campaign from a list
  Given A list of campaigns have been imported from lime
    | Codi campanya | Nom campanya | Codi tipus campanya | Tipus campanya |
  Data extracció Lime | Data importació |
  |101|18-19 Assignatura-professor màsters P 1r S|25|Enquesta assignatura-
  professor grau i màster univ.|2020-11-06|2021-03-15|

  When I select a campaign
  Then I'm viewing a list of comments from the selected campaign
    |sid |Descripció |Tipus preg. |Codi pregunta |Pregunta |tid |Idioma
  |Comentari original |Tipus incidència |Solució |Comentari proposat |
    |916267 |Enquesta als titulats/des en MÀSTER DISSENY I GESTIÓ D'ENTORNS
  BIM: EDIFICACIÓ, ESTRUCTURES I INSTAL·LAC - 034M ||PR020 |Punts forts /
  Àrees de millora | 8 | ca |Millorar en l'organització del curs, i el
  compliment dels horaris.||||

Scenario: Try to see list of campaigns but not logged in
  Given I'm not logged in
  When I list the campaigns
  Then I'm redirected to the login form

```

## Feature: View survey comment

```

Feature: View survey comment
In order to show the detail of a comment associated with a survey
As a user
I want to see the details of the comment

Background: There is a registered user
  Given Exists a user "user" with password "password"
  And I login as user "user" with password "password"

Scenario: Select a comment from a list
  Given A set of surveys and comments have been imported from Lime
  And I'm viewing the list of surveys and comments
    | sid | Descripció | Tipus pregunta |Codi pregunta | Pregunta | tid
  | Idioma | Comentari original | Tipus incidència | Comentari proposat |
    |916267 |Enquesta als titulats/des en MÀSTER DISSENY I GESTIÓ
  D'ENTORNS BIM: EDIFICACIÓ, ESTRUCTURES I INSTAL·LAC - 034M ||PR020 |Punts
  forts / Àrees de millora | 8 | ca |Millorar en l'organització del
  curs, i el compliment dels horaris.||||

  When I select a comment
  Then I see the details of the comment in an editable form
    | Tipus pregunta | Codi pregunta | Tid | Pregunta | Comentari original
  | Idioma | Tipus incidència | Proposta solucio | Comentari proposat |

```

## Feature: Edit Comment

**Feature:** Edit Comment

In order to edit the content of a comment

As a user

I want to see the details of the original commentary, including type of issue, type of proposed solution and the proposed commentary.

It must be possible edit all the field unless the original commentary

**Background:** There is a registered user

**Given** Exists a user "user" with password "password"

**And** I login as user "user" with password "password"

**Scenario:** Select a comment from a list

**Given** A record of comment have been imported from Lime

**And** I'm viewing the fields of the comment

| sid | Descripció | Codi pregunta | Pregunta | tid | Comentari original | Tipus incidència | Comentari proposat | Acceptar |

**When** I modify the field Tipus incidència

**And** I modify the field Proposta solucio

**And** I modify the field Comentari proposat

**And** I accept the changes

**Then** The fields modified have been saved

**And** the program returns to comments page

## Feature: Import surveys

**Feature:** Import surveys

In order to review comments in the surveys of a campaign

As a user

I want to import surveys and comments of a campaign from Lime

**Background:** There is a registered user

**Given** Exists a user "user" with password "password"

**And** I login as user "user" with password "password"

**Scenario:** Import surveys and comment of a campaign

**Given** A list of campaigns have been imported from lime

| Codi campanya | Nom campanya | Codi tipus campanya | Tipus campanya | Data extracció Lime | Data importació |

|101|18-19 Assignatura-professor màsters P 1r S|25|Enquesta assignatura-professor grau i màster univ.|2020-11-06|2021-03-15|

**When** I click the import button from a campaign

**Then** Surveys of the campaign are imported from Lime

**And** Professor of the imported surveys are imported from Lime

**And** Comments of the imported surveys are imported from Lime

**And** Import date is updated

**And** I'm viewing the list of comments imported from Lime of the selected campaign

|sid |Descripció |Tipus preg. |Codi pregunta |Pregunta |tid |Idioma |Comentari original |Tipus incidència |Solució |Comentari proposat |

|916267 |Enquesta als titulats/des en MÀSTER DISSENY I GESTIÓ D'ENTORNS BIM: EDIFICACIÓ, ESTRUCTURES I INSTAL·LAC - 034M ||PR020 |Punts forts / Àrees de millora | 8 | ca |Millorar en l'organització del curs, i el compliment dels horaris.||||



## Feature: Process comments

**Feature:** Process comments

In order to analyze the classification of the comment in a campaign

As a user

I want to process the comments of the campaign with the NLP Spacy model

**Background:** There is a registered user

Given Exists a user "user" with password "password"

And I login as user "user" with password "password"

**Scenario:** Process comments of a campaign

Given A list of comments of a campaign have been imported from lime

|sid |Descripció |Tipus preg. |Codi pregunta |Pregunta |tid |Idioma

|Comentari original |Tipus incidència |Solució |Comentari proposat |

|916267 |Enquesta als titulats/des en MÀSTER DISSENY I GESTIÓ D'ENTORNS BIM: EDIFICACIÓ, ESTRUCTURES I INSTAL·LAC - 034M ||PR020 |Punts forts / Àrees de millora | 8 | ca |Millorar en l'organització del curs, i el compliment dels horaris.||||

When I click the button process comments

Then Processing of comments starts with the NLP Spacy model

And A timer starts counting 5 second

And An information panel is visible with information about the processing state

And I'm viewing the list of comments of the campaign

|sid |Descripció |Tipus preg. |Codi pregunta |Pregunta |tid |Idioma

|Comentari original |Tipus incidència |Solució |Comentari proposat |

|916267 |Enquesta als titulats/des en MÀSTER DISSENY I GESTIÓ D'ENTORNS BIM: EDIFICACIÓ, ESTRUCTURES I INSTAL·LAC - 034M ||PR020 |Punts forts / Àrees de millora | 8 | ca |Millorar en l'organització del curs, i el compliment dels horaris.||||

**Scenario:** Comments are being processed

Given A list of comments of a campaign have been imported from lime

|sid |Descripció |Tipus preg. |Codi pregunta |Pregunta |tid |Idioma

|Comentari original |Tipus incidència |Solució |Comentari proposat |

|916267 |Enquesta als titulats/des en MÀSTER DISSENY I GESTIÓ D'ENTORNS BIM: EDIFICACIÓ, ESTRUCTURES I INSTAL·LAC - 034M ||PR020 |Punts forts / Àrees

When Timer is activated

And Processing of comments is executing

Then Information panel is updated with processing state

And Timer counts 5 seconds more

And I'm viewing the list of comments of the campaign

|sid |Descripció |Tipus preg. |Codi pregunta |Pregunta |tid |Idioma

|Comentari original |Tipus incidència |Solució |Comentari proposat |

|916267 |Enquesta als titulats/des en MÀSTER DISSENY I GESTIÓ D'ENTORNS BIM: EDIFICACIÓ, ESTRUCTURES I INSTAL·LAC - 034M ||PR020 |Punts forts / Àrees de millora | 8 | ca |Millorar en l'organització del curs, i el compliment dels horaris.||||

**Scenario:** Processing has finished

Given A list of comments of a campaign have been imported from lime

|sid |Descripció |Tipus preg. |Codi pregunta |Pregunta |tid |Idioma

|Comentari original |Tipus incidència |Solució |Comentari proposat |

|916267 |Enquesta als titulats/des en MÀSTER DISSENY I GESTIÓ D'ENTORNS BIM: EDIFICACIÓ, ESTRUCTURES I INSTAL·LAC - 034M ||PR020 |Punts forts / Àrees

When Timer is activated

```
And Processing of comments has finished
Then Information panel show the final of the process
And Closed button is visible
And I'm viewing the list of comments of the campaign
|sid |Descripció |Tipus preg. |Codi pregunta |Pregunta |tid |Idioma
|Comentari original |Tipus incidència |Solució |Comentari proposat |
|916267 |Enquesta als titulats/des en MÀSTER DISSENY I GESTIÓ D'ENTORNS
BIM: EDIFICACIÓ, ESTRUCTURES I INSTAL·LAC - 034M ||PR020 |Punts forts /
Àrees de millora | 8 | ca |Millorar en l'organització del curs, i el
compliment dels horaris.||||
```

## Results / Conclusions

In this iteration, the original application has been identified from where data will come and, the structure of this database has been explored.

It also has been defined and created the database of this project implemented in MySQL.

Following, it has been defined the features and scenarios that will be implemented in this project.

And finally, a prototype of the application has been implemented. It consists of html pages with simulated information and a simple interaction between pages.

## Iteration 2. Sample file load and language detection

### Requirements / Specification

As a preliminary step to the classification of survey comments using Natural Language Processing, it is necessary to load the information from original sample comments and adapt its structure to make it useful as a data set training and testing of the models to be generated.

For the loading of the sample data we face another problem, the source files do not specify the language in which they have been entered, being able to be in Spanish, Catalan or English. At the time of processing the files, the language of each comment must be detected, as each language must be processed separately.

The original data are Excel files with a uniform column structure, as they are extracted from the LimeSurvey survey management application.

The original files are organized into folders, with the following structure: *Original / Curs / Semestre / Arxiu*

But this structure is not fixed and there may be files with different format and content than desired. The tool must be able to go through the entire folder structure and validate the files that have the correct structure.

The following picture shows an example of the folder structure:



Image 5 Example of folder structure

This would be an example of sample data:

1	A	B	C	D	E	F	G	H	I	J	K	M	N	O	P	Q
	Curs	Quadr	Codi	Estudi	Codi assign	Assignatura	Codi	Grup	Codi pregu	Pregunta	Codi	Co gn	Co gn	No m	Comentari	Tipus d'incidència
2	2017-18	1	M19	MU en Llengües Aplicades	12355	INTRODUCCIÓ A LA RECERCA CIENTÍFICA	1	Grup d'Introducció a la recerca	A02	COMENTARIS: (punts forts / àrees de millora)	0				buenos foros y disponibilidad de los profesores para responder preguntas.	
3	2017-18	1	M19	MU en Llengües Aplicades	12355	INTRODUCCIÓ A LA RECERCA CIENTÍFICA	1	Grup d'Introducció a la recerca científica	P0201	COMENTARIS: (punts forts / àrees de millora)	1	Xxxxx	Xxxxx	Xxxxx	Módulo 1 áreas para mejorar: _x000D_ La tarea nr. 1 del módulo 1 buscaba corregir un artículo científico. Sin embargo, considero que hay que estar un poco en la cabeza de la profesora para entender lo que buscaba. La corrección y comentarios son vagos. No me pareció una actividad muy constructiva._x000D_ Módulo 2-5 excelentes profesores	Passar a assignatura
4	2017-18	1	M19	MU en Llengües Aplicades	12355	INTRODUCCIÓ A LA RECERCA CIENTÍFICA	1	Grup d'Introducció a la recerca	P0202	COMENTARIS: (punts forts / àrees de millora)	2	Yyyyy	Yyyyy	Yyyyy	La profesora del primer módulo podría mejorar su forma de hacer comentarios.	
5	2017-18	1	M31	MU en Sistema de Justicia Penal	13300	DOGMÀTICA PENAL I POLÍTICA CRIMINAL	1	DOGMÀTICA PENAL I	A02	COMENTARIS: (punts forts / àrees de millora)	0				Asignatura muy bien planteada y prácticas retadoras	
6	2017-18	1	M31	MU en Sistema de Justicia Penal	13300	DOGMÀTICA PENAL I POLÍTICA CRIMINAL	1	DOGMÀTICA PENAL I	P0203	COMENTARIS: (punts forts / àrees de millora)	3	Zzzzz	Zzzzz	Zzzzz	Gran profesor	
7	2017-18	1	M31	MU en Sistema de Justicia Penal	13300	DOGMÀTICA PENAL I POLÍTICA CRIMINAL	1	DOGMÀTICA PENAL I	P0204	COMENTARIS: (punts forts / àrees de millora)	4	Vvvvv	Vvvvv	Vvvvv	No he tenido contacto	No ha dado clase
8	2017-18	1	M31	MU en Sistema de Justicia Penal	13301	DELICTES CONTRA BÉNS JURÍDICS INDIVIDUALS I	1	DELICTES CONTRA BENS	P0201	COMENTARIS: (punts forts / àrees de millora)	1	Xxxxx	Xxxxx	Xxxxx	No he tratado con él	No ha dado clase
9	2017-18	1	M31	MU en Sistema de Justicia Penal	13301	DELICTES CONTRA BÉNS JURÍDICS INDIVIDUALS I	1	DELICTES CONTRA BENS	P0202	COMENTARIS: (punts forts / àrees de millora)	2	Yyyyy	Yyyyy	Yyyyy	Muy buenas clases	
10	2017-18	1	M31	MU en Sistema de Justicia Penal	13301	DELICTES CONTRA BÉNS JURÍDICS INDIVIDUALS I	1	DELICTES CONTRA BENS	P0204	COMENTARIS: (punts forts / àrees de millora)	4	Vvvvv	Vvvvv	Vvvvv	No he tratado con él	No ha dado clase
11	2017-18	1	M37	MU en Estudis de Gènere i Gestió de Polítiques d'Igualtat	14643	POLÍTIQUES PÚBLIQUES I GÈNERE	1	POLÍTIQUES PÚBLIQUES I	A02	COMENTARIS: (punts forts / àrees de millora)	0				La profesora no ha colgado unidades didácticas cosa que ha dificultado el aprendizaje. No había material adecuado	Passar a professor
12	2017-18	1	M37	MU en Estudis de Gènere i Gestió de Polítiques d'Igualtat	14643	POLÍTIQUES PÚBLIQUES I GÈNERE	1	POLÍTIQUES PÚBLIQUES I	A02	COMENTARIS: (punts forts / àrees de millora)	0				Es fan els exercicis, sense cap mena d'implicació ni explicació per part de la professora i no ens dona ni tan sols el resultat....en fi...sin palabras	Passar a professor

Image 6 Example of sample data

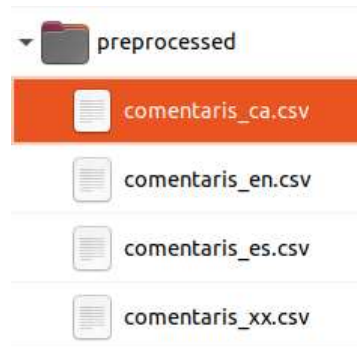
The file prepared for the training of the model must be in csv format and it must have the structure of the following example:

1	A	B	C	D	G
	Curs	TipusPregu	TipusIncidència	Comentari	Idioma
16	2017-18	A	Comentari de professor	Es fan els exercicis, sense cap mena d'implicació ni explicació per part de la professora i no ens dona ni tan sols el resultat....en fi...sin palabras	ca
20	2017-18	A	Comentari de professor	cal més feedback amb el professor i millor comunicació entre professor i alumnes.	ca
27	2017-18	P	Comentari d'assignatura	No he après res, segueixo sense saber l'ati.	ca
34	2018-19	P	No ha impartit classe a aquest grup	No he tingut aquesta professora.	ca
37	2018-19	P	No ha impartit classe a aquest grup	No l'hem tingut	ca
40	2018-19	P	No ha impartit classe a aquest grup	no l'he tingut	ca
42	2018-19	P	No ha impartit classe a aquest grup	No he tingut aquesta professora	ca

Image 7Example of csv structure

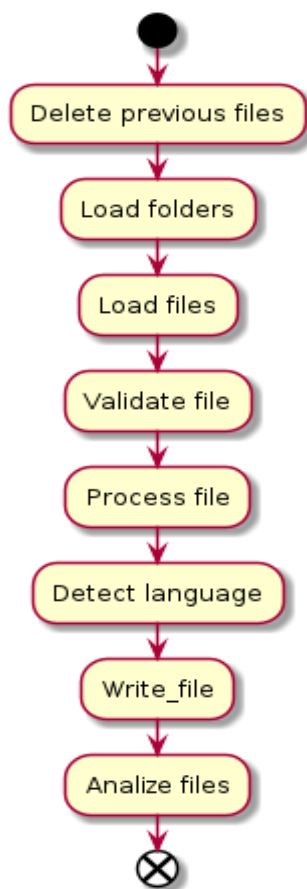
The solution developed must be able to detect the language of comments because the original comments don't include this information. The possible languages are Catalan (ca), Spanish (es) and English (en).

One file will be created per language (ca, es and en) and another for the other languages (xx). The following image shows the result:



*Image 8 Example of resultant files*

## Design



*Schema 7 Activity graph of load\_data*

The initial data is in a single folder and the format of the data has been unified.

The subtasks that are made are:

**Delete previous files:** Delete files from previous executions. In each execution, all the original files will be processed and the processed files will be generated again.

**Load folders:** Recursively traverse the base folder of the original files to treat the contents of each of them.

**Load files:** Scan the files in the current folder to treat them.

**Validate file:** For each file, check that it has Excel format and, for each tab it contains, check that it has predefined columns.

**Process file:** Filter the records needed by the model and select the desired columns.

**Detect language:** For each record in the file, detect the language of the comment. This part will be explained in detail later.

**Write\_file:** Write the prepared data of the current comment in the target processed file, chosen depending on the language.

**Analyze\_files:** To check the correct loading of the data and detect possible incidents, some statistics on the number of cases by language, course, type of question and type of incidence will be extracted.

## Implementation

The following tools shall be used for the implementation of this iteration:

- Python 3.8: General purpose programming language
- Anaconda 3: Programming tool environment
- Jupyter 6.1.4: Tool to generate Notebook scripts, which allow you to create and test code interactively.
- Pandas 1.1.3: For processing both Excel and text data.
- MySQL 8.0.22: For storing the results, use the user interface.
- Spacy v2.3: Natural Language Processing (NLP).
- Spacy.spacy\_langdetect: Spacy comments language detection.
- PyCld2 0.41: Another language detection tool for comments.
- Ccoreilly / spacy-catala

The Notebook in which the loading of the original files has been developed is: **load\_data.ipynb**

Spacy includes models that support Spanish and English languages. And it's capable of detecting the language of a text using the *LanguageDetector* component.

While Spacy doesn't support the Catalan language by default, it's necessary to add an external model that supports it.

To detect and process the Catalan language had been installed the ccoreilly/spacy-catala model:

<https://github.com/ccoreilly/spacy-catala>

This model includes:

- Word vectors of fastText
- Grammar, morphology and syntax, using the information from AnCorra
- To the extraction of entities, it uses annotations from the wikipedia (Cross-lingual Name Tagging and Linking for 282 Languages)

Test of language detection and processing of sentences in Spanish, Catalan and English have been done in the Jupiter Notebooks:

- *Introduction\_lang\_detection.ipynb*
- *Introduction\_ca.ipynb*
- *Introduction\_es.ipynb*
- *Introduction\_en.ipynb*

## Results / Conclusions

After loading sample data, we have just over 7,900 comments to build the training and test sets. The majority of these comments are in Catalan (76%), with a lower percentage in Spanish (17%) and barely comments in English (4%).

I wanted to point out that Spacy is not able to detect the language of a part of comments (188 cases). Although they are few, it can hide a major problem. For this reason, the problem of language detection will be analyzed in the next iteration and some way to improve the detection will be sought.

The following table shows the distribution of the comments uploaded by academic year and language, where the symbol 'xx' represents the languages different of Spanish, Catalan or English.

Academic year	ca	en	es	xx	Total	Percentage
2017-18	27	2	55	6	90	1,14%
2018-19	873	248	297	50	1.468	18,56%
2019-20	5.144	76	1.001	132	6.353	80,31%
Total	6.044	326	1.353	188	7.911	
Percent	76,40%	4,12%	17,10%	2,38%		

Table 1 Distribution of comments by language and year

Of the total of original comments, 43% correspond to questions about the subject and the remaining 57% to evaluations about the teacher. The following table shows the distribution of comments by question type.

Question Type	ca	en	es	xx	Total	Percentage
A	2.633	97	625	59	3.414	43,16%
P	3.411	229	728	129	4.497	56,84%

Table 2 Distribution of comments by question type

Finally, it is interesting to know how many of these comments can be considered an issue and of what type. In total, we have 503 reviews classified by users as an issue, which is only 6% of the total. They are few, but the difficulty of detecting them within the set of surveys is what justifies this project.

Particularly significant is the 'Comentari problemàtic' type, with the 54% of the issues and the types 'No ha impartit docència' and 'Comentari de professor en pregunta d'assignatura' with the 23% and 11% of the cases

Destaca especialmente el tipo 'Comentari problemàtic', con el 54% de las incidencias y los casos de 'No ha impartit docència' y 'Comentari de professor en pregunta d'assignatura' with the 23% and 11% of cases. The following table shows the distribution by type of issue.

Issue type	ca	en	es	xx	Total	%
Canvi de professor	15	0	2	0	17	3,38%
Comentari d'assignatura	3	1	3	0	7	1,39%
Comentari de professor	37	2	13	1	53	10,54%
Comentari excel·lent	1	2	1	0	4	0,80%
Comentari ofensiu	7	0	2	2	11	2,19%
Comentari problemàtic	230	0	39	2	271	53,88%
Duplicar a professor	2	0	3	0	5	0,99%
Exclamacions o emoticones excessius	1	0	0	1	2	0,40%
Faltes d'ortografia	14	0	4	0	18	3,58%
No ha impartit classe a aquest grup	35	33	21	26	115	22,86%
<b>Total</b>	<b>345</b>	<b>38</b>	<b>88</b>	<b>32</b>	<b>503</b>	

*Table 3 Distribution of comments by issue type*

In the special case of 'No ha impartit classe' a very high proportion of comments in language 'xx' has been detected. This case reinforces the need to further analyze language detection.



## Iteration 3. Analysis and improvement of language detection

### Requirements / Specification

As mentioned in the previous section, the language detection of the comments with **LanguageDetector** of Spacy doesn't manage to identify all the cases. For this reason, the **PyCld2** library, specialized in language detection, will be tested and results will be compared with the *Spacy* library.

### Design

The **detect\_language** function will be modified to also use the *PyCld2* library, storing both languages in the processed file.

A new Notebook will read the processed files with the two languages and it will make a comparison to study how to improve detection by combining them. The result will be a new *detect\_language* function that generates a combined third language.

Finally, the *detect\_language* function will be modified again by incorporating the results obtained from the analysis.

### Implementation

The notebook to analyze the problem in language detection is: '**lang\_detect\_problem.ipynb**'

The function '*load\_data*' of this notebook loads the 3 processed files (*comentaris\_xx.csv*) of the languages *ca*, *es*, *en* and *xx*, joining the data in a single *DataFrame* and selecting the columns that interest: *Comment*, *TipusIncidence*, *IdiomaSpacy* and *IdiomaPyCld2*.

Different queries group the information using *Pandas DataFrames* and compare the two languages.

### Results / Conclusions

As a result of the comparison, the following conclusions are drawn:

In general, there is a high level of overlap between the two languages:

- Texts with same language:  $7.351 / 7855 = 93,58 \%$
- Texts with different languages: 504 records. They're candidates for recovery.

The following table shows the comparison between the two languages:

IdiomaSpacy	IdiomaPyCld2	Comments	Percentage
ca	ca	5.839	73,99%
es	es	1.300	16,47%
en	en	232	2,94%
ca	en	140	1,77%
en	ca	57	0,72%
it	ca	24	0,30%
es	ca	20	0,25%
es	en	20	0,25%
pt	es	19	0,24%
fr	ca	16	0,20%

*Table 4 Comparative between Spacy and PyCld2*

Review of Catalan language:

Texts detected as Catalan actually are in English.

- The texts detected as ca (spacy) - en (pycld2) are actually in Catalan (141 cases)
  - The texts detected as ca (spacy) - es (pycld2) are actually in Catalan (11 cases)
  - The texts detected as ca (spacy) - xx (pycld2) are actually in Catalan (31 cases)
- Conclusion: In general, the detection of Catalan with Spacy is correct.

Review of the Spanish language:

- The texts detected as es (spacy) - ca (pycld2) are actually in Catalan (16 cases)
  - The texts detected as es (spacy) - en (pycld2) are actually in Spanish (19 cases)
  - The texts detected as es (spacy) - xx (pycld2) are actually in Spanish (13 cases)
- Conclusion: In general, the detection of the Spanish language with Spacy is correct, except for the Catalan language, which Pycld2 could improve.

Review of English language:

- The texts detected as en (spacy) - ca (pycld2) are actually in Catalan (60 cases)
  - The texts detected as en (spacy) - es (pycld2) are actually in Spanish (8 cases)
  - The texts detected as en (spacy) - xx (pycld2) are actually in Catalan (25 cases)
- Conclusion: In the case of Spanish and Catalan languages, the detection of Pycld2 is better, and in the case of other languages, we should make a direct assignment to Catalan.

Review of other languages (xx represents languages that are not ca, es, en):

- The texts detected as xx (spacy) - ca (pycld2) are actually in Catalan (68 cases)
- The texts detected as xx (spacy) - es (pycld2) are actually in Spanish (28 cases)

- The texts detected as xx (spacy) - en (pycld2) are actually in Catalan (30 cases)
- The texts detected as xx (spacy) - xx (pycld2) are almost all Catalan and the rest Spanish (55 cases)

Conclusion: For the Spanish and Catalan languages, the detection of Pycld2 is better, and in the case of other languages, we should make a direct assignment to Catalan.

The following table shows the number of cases for each language combination and the resulting language proposal. The red labels indicate that the Spacy language has to be changed.

Language Spacy	Language Pycld2							
	ca		es		en		xx	
ca	ca	5.815	ca	11	ca	141	ca	31
es	ca	16	es	1.297	es	19	es	13
en	ca	60	es	8.0	en	239	ca	25
xx	ca	68	es	28	ca	30	ca/es	55

Table 5 Crosstable of languages in Spacy and PyCld2

Grouping by type of incidence:

- Of the 505 recoverable records, 74 are issues.
- Most occurrences (59 records) are of 'No ha impartit classe' type.
- In the entire data set, there are 440 incidents, 37 of type 'No ha impartit classe'. Improved language identification would be of particular benefit to this type.

The following table shows the comments with different language in Spacy and PyCld2 grouped by type of incidence:

Issue type	
Comentari de professor	2
Comentari excel·lent	1
Comentari ofensiu	4
Comentari problemàtic	7
Exclamacions o emoticones excessius	1
No ha impartit classe a aquest grup	59
<b>Total</b>	<b>74</b>

Table 6 Distribution of comments without language

## Iteration 4. Model 1. Issue ‘No ha impartit classe’

### Requirements / Specification

In this second iteration, the initial functionalities of Natural Language Processing will be implemented to classify the comments. Different models will be created that will be trained using different labeled comment sets. These comments come from the surveys of the courses 2017-18, 2018-19 and 2019-20.

The aim is to label the comments according to the following classifications:

- ‘No ha impartit classe a aquest grup’
- ‘Comentari d’assignatura’
- ‘Comentari de professor’
- ‘Faltes d’ortografia’
- ‘Exclamacions o emoticones excessius’
- ‘Comentari problemàtic’
- ‘Comentari ofensiu’

A different model will be created for each type of incidence. The data set needed to train each model may be different. For example, the first category affects only teacher question comments, the second category only subject type questions, misspellings will be detected with a different type of tool, ...

This iteration shall address the type of incidence: ‘No ha impartit classe a aquest grup’.

### Design

A loop will load the processed data, train and evaluate, creating a separate model for each language.

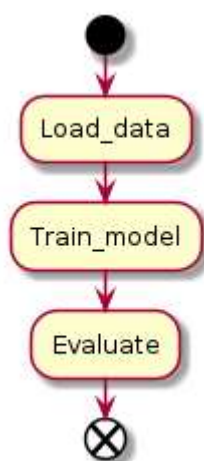


Table 7 Activity graph of  
Train model 1

The tasks to be performed for each model are:

**Load\_data:** Read the processed data of the language to work. The result is a set of data for process the training and another for the evaluation. 20% of the comments will be reserved for the evaluation. In addition, if the number of comments contained in the training set has to be restricted, we should do this so that all positive cases are included and the negative cases are limited.

**Train\_model:** The training and test sets created in the above process will be used to create the model. This model will be saved to disk and evaluated. Since few positive data are available in the test set, we will try to optimize the examples by rearranging them randomly, selecting subsets and doing several iterations.

**Evaluate:** For each comment in the evaluation set, the model will obtain its prediction. This prediction will be compared with the label assigned by the user. In this way, the true and false positives and the true and false negatives will be obtained. The indicators to assess the results will be Precision (P), Recall (R) and F-score.

## Implementation

The following tools shall be used for the implementation of this iteration:

- Python 3.8: General programming language.
- Anaconda 3: Programming toolkit environment.
- Jupyter 6.1.4: Tool to generate Notebook scripts that allow generating and testing code interactively.
- Pandas 1.1.3: Library for data processing both Excel and text files.
- MySQL 8.0.22: Database server used to save the results that will be used in the user interface.

The classification implementation of the issue 'No ha impartit classe' is based on the **TextCategorizer** component of *Spacy*.

The Notebook in which this part is implemented is '**model\_no\_classe\_lang\_2.ipynb**'. Some details are described below.

## Load\_data

The **load\_data** function loads the csv containing the original processed comments corresponding to the selected language. For example, 'comentaris\_ca.csv'. Next, transform the information to the desired format and divide it into training and test sets. Specifically, the steps it takes are:

- Load the csv with the processed comments and convert it into a *DataFrame* of *Pandas*.
- Select comments of type '*Professor*' (*TipusPregunta* = 'P'), and fields '*Comentari*' and '*TipusIncidencia*'.
- Transforms the field '*TipusIncidencia*' with value '*No ha impartit classe a aquest grup*' in a tuple with the format: ([*Comentari*], {"POSITIVE": True, "NEGATIVE": False}), or the inverse option if it hasn't this value.
- Split the tuple in two lists: text list and category list.
- From there divide each list into a part for training and another for evaluation, taking into account that the function has been passed as parameter the number of texts that can contain at most the training set. By default it will use 80% of the comments for training and 20% for testing, as well as a maximum of 2,000 comments in the training set.

- Since there are few positive cases, compared to the rest, I decided to use them all, dividing them proportionately between training and test. For this reason, the maximum should be applied to the number of negative texts.

This function returns four lists:

- Training texts
- Training categories
- Test texts
- Test categories

The following graph represents the grouping of data into a training set and a test set, distinguishing between positive and negative cases. It has been assumed that a total of 4,000 original comments are available, with a high disproportion between positive and negative cases and a limit of 2,000 comments on the number of comments to use for training.

Original data:	4.000	Train (80%)		Test (20%)
Positive:	75	60		15
Negative:	3.925	Discarded (1.200)	Limit (1.940)	785
Total:	2.800	1.200	2.000	800

Table 8 Distribution of comments in Training and Test datasets

### Train\_model

This is the function that takes care of preparing the sample data, training the model and evaluating it. The steps you do are:

- It loads the *Spacy* model you want to train, depending on the language. For example, the Catalan model is: **ca\_fasttext\_wiki**
- The **TextCategorized** component is added to the pipeline, with the key '*textcat*', and the labels to use for positive and negative cases are defined ('POSITIVE' y 'NEGATIVE').
- The **load\_data** function is called to return the four lists indicated above: Training Texts, Training Categories, Test Texts and Test Categories. This indicates the number of maximum texts for the training, the ratio between the training data and test data and the language to load.
- The data of each element of the training set is transformed to have the format: (text, {'cats': {'POSITIVE': True, 'NEGATIVE': False}})
- Training begins with the function **nlp.begin\_training()**.
- To optimize the available training comments, the order of comments is changed with the **shuffle** function and subsets of data (batches) are created with the **Compounding**

and

### **Minibatch**

components.

- Training is done, with the function **nlp.update**.
- The model is evaluated using the evaluate function, which will be explained later and which returns the indicators **Precision** (*p*), **Recall** (*p*) and **F\_score** (*f*).
- To improve the detection capacity of the model, the training is repeated by means of a loop, using different data sets (**batches**).
- To determine the number of loops required, the function uses the **loss** value, returned by the training, which indicates the difference between the user-defined labels and the model prediction. This value tends to decrease as the model is better trained and the loop continues until it reaches a minimum value, default 0.001. This is usually achieved in 5 or 6 turns of the loop. A maximum of 20 loops is also defined.
- Once trained, the model is saved to disk for later use, using the **nlp.to\_disk method**.
- Finally, the model is tested by calculating the prediction for a few fixed cases. This is done with the model loaded in memory and with the model saved on disk.

### **Evaluate**

This function evaluates the model results, calculates the prediction for each comment and compares it with the user-defined label. As a result, calculate the indicators *Precision* (*p*), *Recall* (*r*) and *F-score* (*f*).

Four types of cases are defined:

- True positives (TP): The result of the prediction and the user label are equal and positive.
- False positives (FP): The prediction result is positive, but the user label is negative.
- True negatives (TN): The result of the prediction and the user label are equal and negative.
- False negatives (FN): The prediction result is negative, but the user label is positive.

The following confusion table shows the relationship between the different types.

		User label	
		Positive	Negative
Pre dic tion	Positive	TP	FP
	Negative	FN	TN

*Schema 8 Table of confusion*

A loop calculates the prediction of all test cases and calculates how many of them are included in each case.

From the above result the following indicators are calculated:

- Precision (P) = True positives (TP) / (True positives (TP) + False positives (FP))

It indicates the accuracy in the detection of positive cases. That is, the extent to which positive cases are defined as not positive.

The following table represents the calculation of this indicator:

		User label	
		Positive	Negative
Pre dic tion	Positive	TP	FP
	Negative	FN	TN

*Schema 9 Calculation of Precision (P) indicator*

- Recall (R) = True positives (TP) / (True positives (TP) + False negatives (FN))

It indicates the ability to detect positive cases. That is, how many of all the positives are able to identify the model.

The following table shows the calculation of the indicator:

		User label	
		Positive	Negative
Pre dic tion	Positive	TP	FP
	Negative	FN	TN

*Schema 10 Calculation of Recall (R) indicator*

- F-score (F) =  $2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall}) = 2 * \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} * \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$

It shows the combined effect of the above two indicators, as can be seen in the following table:



		User label	
		Positive	Negative
Pre dic tion	Positive	2 x TP	FP
	Negative	FN	TN

Schema 11 Calculation F-score (F) indicator

## Results / Conclusions

In the **load\_data** step you get the following data sets per language:

language	Total data	Train	Train true	Train false	Test	Test true	Test false
Catalan	3.955	<b>2.000</b>	58	1.942	<b>791</b>	15	776
Spanish	942	<b>753</b>	30	723	<b>188</b>	8	180
English	147	<b>117</b>	3	114	<b>29</b>	1	28

Table 9 Distribution of data set by language and type

The main peculiarity of the data load is that all positive data are always used. On the other hand, the limitation in the number of registers only applies to the Catalan language since the rest have few examples.

The main peculiarity in the **train\_model** step is that instead of executing a fixed number of iterations, they continue running until losses reach a minimum level (0.001). This level is usually achieved with 5 or 6 iterations. **Compounding**, **minibatch** and **shuffle** tools are used to create subsets of data (batches) in random order and optimize the use of sample data.

Spacy has several models for each language. In the **evaluation** step the training has been tested for each of them. The intention is to find out if the model selection affects the result.

For each language and model, the training was performed 5 times and the results were averaged.

Language	Model	Precision (p)	Recall (r)	F-score (f)
Catalan	ca_fasttext_wiki	0,8896	0,7468	0,8058
Catalan	ca_fasttext_wiki_lg	0,9418	0,6666	0,7722
Spanish	es_core_news_sm	0,9750	0,6000	0,7250
Spanish	es_core_news_md	0,9094	0,6750	0,7704
Spanish	es_core_news_lg	0,9142	0,7500	0,8228
English	en_core_web_sm	0,0000	0,0000	0,0000
English	en_core_web_md	0,0000	0,0000	0,0000
English	en_core_web_lg	0,0000	0,0000	0,0000

Table 10 Performance by language and model

There is no clear improvement in the results by using a more complete model in the case of the Catalan language. But in the case of Spanish language you can see a difference.

The number of examples of Catalan seems sufficient to use the model created.

The number of examples in the case of Spanish seems insufficient, although it achieves correct results. The prediction should be reinforced with some other technique.

The English language does not have enough examples to train and evaluate the model.

The **evaluate** function also returns the list of false positive or false negative cases, so you can manually check if the prediction is justified. Below you can see some of them:

False positives:

Catalán:

- *"No l'hem tingut a aquesta persona (Karen), hem tingut al Marc a història."*
- *"El millor professor que he tingut."*
- *"La professora Cirera no ha vingut a fer-nos les classes d'aquesta assignatura que estaven programades."*

Spanish:

- *"No se puede opinar de clases que no se han dado."*

False negatives:

Catalan:

- *"No hi ha hagut docència virtual per la seva part. ha penjat un powerpoint en pdf al campus."*
- *"Sincerament, si no ens han donat una teoria concreta no puc evaluar si és o no bona professora."*
- *"Només va venir un dia a la sortida de camp."*

Spanish:

- *"No se sabe nada de él durante la cuarentena."*
- *"No hemos podido conocer a esta profesora. Sin embargo, se ha mantenido en contacto con nosotras por correo electrónico."*

Looking at the cases you can see that they are dubious even for one person and some of them seem to be mislabeled by the user. In my opinion the results of the prediction are more correct than what the indicators show.

## Iteration 5. Model 1 alternative: Parts-of-speech and Dependency parsing

### Requirements / Specification

In the above iteration, the **TextCategorizer** component has been used to create a model to identify comments that respond to the type of incident: *'No ha impartit classe'*.

The model achieves good results in Catalan and Spanish. However, in the latter language the low number of cases suggests that another method should be used to classify this type of comment.

A better classification method might be using the pre-trained models available in *Spacy* and using the **Part-of-speech** and **Dependency parsing** methods to identify the different components of the phrases and their dependencies. Then we can use the **Rule-based matching** method, specifically **Token-based matching**, to define patterns that can be applied to phrases to decide whether they meet these patterns.

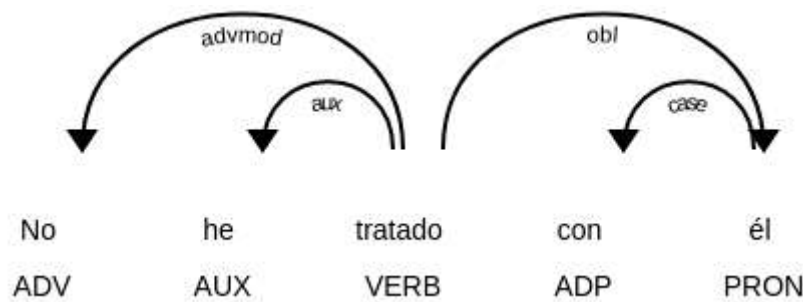
When a text is passed to *Spacy* it is broken down into words (token) and phrases (sents). For each token it uses the statistical models it includes, its definition and its context to obtain information about its morphology and its syntax that it saves as token attributes.

Below we can see some of the information obtained for the sentence: *'No he tratado con él'*.

TOKEN	LEMMA	POS	TAG	DEP	SHAPE
No	No	ADV	ADV__Polarity=Neg	advmod	Xx
he	haber	AUX	AUX__Mood=Ind Number=Sing Person=1 Tense=Pres VerbForm=Fin	aux	xx
tratado	tratar	VERB	VERB__Gender=Masc Number=Sing Tense=Past VerbForm=Part	ROOT	xxxx
con	con	ADP	ADP__AdpType=Prep	case	xxx
él	él	PRON	PRON__Case=Acc,Nom Gender=Masc Number=Sing Person=3 PronType=Prs	obl	xx

Table 11 Example of Parts-of-speech

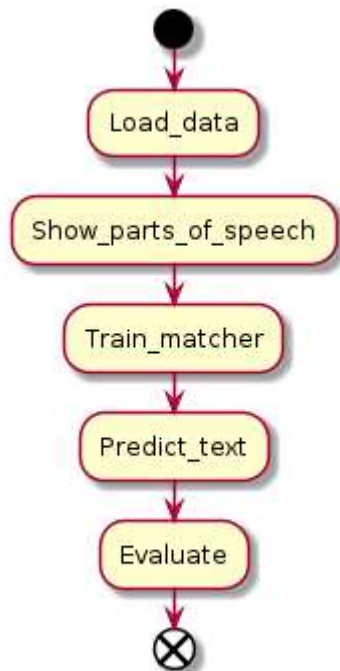
And to facilitate the understanding of the information, *Spacy* provides a graphical representation using the **displaCy visualizer** component.



Schema 12 Dependency parsing

## Design

The following steps will be done for the creation and evaluation of the new model:



Schema 13 Activity graph model 1 - parts-of-speech

**Load\_data:** It will be carried out the reading of the data of the preprocessed files, filtering records corresponding to teacher questions, labelling of incidences of the type '*No impartit class*' and selection of the column's '*comment*' and the '*label*'.

**Show\_parts\_of\_speech:** This function will be used to analyze the desired phrases to identify the characteristics that later help create the patterns to be used in the matching. A comment will be divided into phrases, and in each sentence, the attributes of the tokens and the chart of dependencies will be indicated.

**Train\_matcher:** The patterns that will be used in the next step will be created from the characteristics identified in the previous step. For example, they can be conditions of the type that the phrase begins with the word 'No', followed by a particular verb, and ends with a name of type '*clase*'.

**Predict\_text:** The patterns defined in the previous step will be applied to the sentences contained in the analyzed comment. The comment will be classified as included in the issue type 1

if o any of its sentences matches the pattern.

**Evaluate:** To evaluate the proper functioning of the defined patterns, all comments will be reviewed, calculating the prediction for each of them and comparing with the label specified by the user. As was done in the previous iteration, the sets of true positives, false positives, true negatives and false negatives will be defined. And from these sets the Precision (p), Recall (r) and F-score (f) will be calculated.

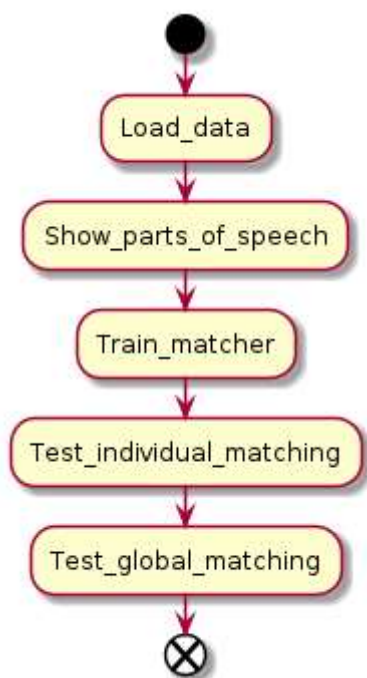
## Implementation

The notebooks in which this new iteration is implemented are:

- **model\_1\_no\_classes\_part-of-speech\_train.ipynb:** This notebook is used to manually analyze the composition of comments labeled positive and define possible patterns. Then the matching is applied to all positive cases to have a first view of the results.
- **model\_1\_no\_classes\_part-of-speech\_evaluate.ipynb:** All sample comments are loaded into this notebook. Each of them is matched with the previous patterns and the result is evaluated. With this the indicators defined in the design are obtained.

-

## Model\_1\_no\_classes\_part-of-speech\_train.ipynb



Schema 14 Activity graph part-of-speech training

The functions created within this notebook are not intended to always run sequentially, but as tools to run by parts and test the results obtained. However, a sequential execution also provides information on the set of positive comments.

The functions included in this notebook will be discussed below:

### Load\_data:

This function loads the data from the pre-processed file of the desired language, in this case Spanish (**comentaris\_es.csv**). To load the data use Pandas and store it in a *DataFrame*.

It filters the data of the '**Teacher**' type questions that have been labeled with '**No ha impartit classe**' type issue and select only the '*Comentari*' column

Group and count comments to remove duplicates and sort by number of occurrences when analyzing them.

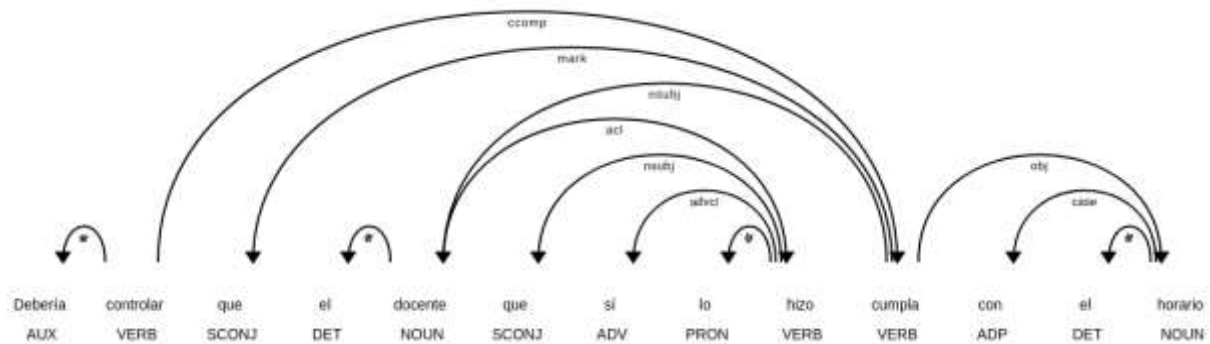
Returns a list of comments sorted by number of occurrences.

### Show parts\_of\_speech:

This function divides the text into sentences and displays for each sentence token some of the attributes that describe the token:

POS:	Identifies the type of word. For example: NOUN, VERB, PRONoun, ...
TAG:	Get more detailed information about the word. For example: gender, number, verbal time, person, etc.
LEMMA:	It contains the root or lexeme of the word, which is the invariable part of the word. For example: from the verb 'Conocemos' we would have 'conocer'.
DEP:	Shows the relationship and dependency between tokens. For example, which is the main verb, which the subject, which complement, which name an adjective refers to,...

It also shows the graphical representation of the relationship between the tokens using the **displaCy** tool. It can analyze and represent complex phrases.



Schema 15 Dependency parsing in training model

## Train matcher

This function uses a Spacy tool called **Matcher** that allows you to define rules similar to regular expressions, but based on the grammar, morphology and semantics of the phrase tokens. Each of the defined rules is called **Patterns**.

This function creates two patterns, one for transitive verbs (those that need an object as a complement) and nontransitive verbs. The list of transitive and nontransitive verbs related to the concept of teaching is previously defined.

The rule for **nontransitive** verbs is: That the phrase begins with a negative adverb, then contain a verb from the defined list. Between the adverb and the verb there can be several words.

The rule for **transitive** verbs is: That the phrase begins with a negative adverb, containing an intransitive verb, from the list, and then contains a name from among the possible ones to identify a class.

Here is the code to define the transitive phrase pattern:

```
# Example: "No ha realizado clases"
pattern = [{"POS": "ADV", "LOWER": {"IN": adv_negs}}, {"OP": "*"}, {"DEP": "ROOT", "POS": "VERB", "LEMMA": {"IN": verbs_trans}}, {"OP": "*"}, {"POS": "NOUN", "LEMMA": {"IN": noms_trans}}]
```

The function returns the *Matcher* object.

## Test individual matching

This function checks if a text meets any of the patterns defined in the *Matcher*. To do this, it breaks down the text into phrases and for each phrase looks for patterns that match, returned as a list of matches. If the text has a match in any of its phrases it is considered to meet the conditions.

This function can be used both to test sample phrases or any phrase you want to analyze to improve matching.

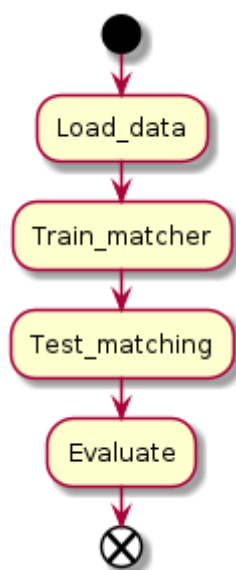
## Text global matching

In this function the list of all positive comments is traversed and calculated if they meet an individual matching. The purpose of the function is to check how many of the examples meet the patterns to try to improve them.

### Model\_1\_no\_classes\_part-of-speech\_evaluate.ipynb

This notebook is based on the previous one, but adapting the code to calculate the matching of all the sample comments and evaluate the model result.

Specific aspects of this notebook will be discussed below.



Schema 16 Activity graph  
part-of-speech evaluate

#### Load data

As in the previous case, comments in the file **comentarios\_es.csv** are loaded. In this case all the comments of the teacher are loaded, not only those of the type of issue desired 'No ha impartit classe'. Comments of the issue type are labeled positive. Returns a DataFrame with the **Comentari** and **Label** columns.

#### Train matcher

Use the same patterns defined in the previous Notebook.

#### Test matching

Check for a given text if any of its phrases meet any of the patterns. Follow the same implementation as the previous notebook.

#### Evaluate

This function calculates the test matching for all comments loaded and compares the user-defined label with the test result.

In this way it counts how many cases there are of each combination: true positives, false positives, true negatives and false negatives.

From these counts the function calculates the indicators Precision (p), Recall (r) and F-score (f), following the same calculations of the previous iteration

## Results / Conclusions

In this new iteration, an alternative way of predicting the classification of a comment as an advocacy member has been implemented. This form consists of defining patterns using the morphology and syntax of the words in the comment and checking if the comments meet these patterns.

In the first notebook a set of functions has been defined that allow studying the structure of sentences, defining possible patterns and testing whether specific phrases match these patterns. These functions can be used autonomously and thus improve the defined patterns.

The second notebook reorganizes the functions of the first to be able to predict the classification of all the comments and implements an evaluation function that allows to calculate different indicators.

The obtained results are:

<b>Precision (p)</b>	<b>Recall (r)</b>	<b>F-score (f)</b>
37,50%	78,95%	50,84%

Due to the system used in the implementation of the patterns the model is able to detect a high percentage of positive cases (recall). However, the patterns are so general that they generate a high number of false positives, so the accuracy is low. Overall, the result is only acceptable, according to the f-score indicator.

Compared with the results of the above iteration, this method is not as accurate. However, it has the possibility of improving the accuracy by better analyzing false positives and adjusting the patterns to discard them.



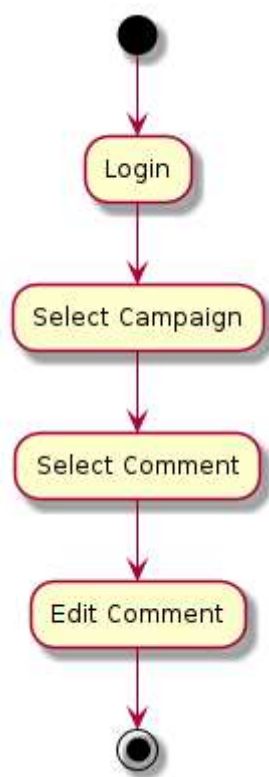
## Iteration 6. User interface implementation

### Requirements / Specification

In the first phase of implementation of the user interface, it was defined the design of this interface, it was created the database and it was implemented the prototype of the screens. Scenarios for the test plan were also defined following the Agile Behaviour Driven Development (BDD) methodology.

In this second phase of implementation, the basic functionalities of the user interface will be programmed starting from a test data load.

A third phase will process the data load from the actual source environment to the application being developed.



*Schema 17 Activity graphs user interface*

### Design

As already explained in the first phase the Django programming environment is used for all the facilities it provides.

The main functionalities or screens of the application and the interaction between them will be developed:

- **Login:** The system will ask for the user and password to access the application.
- **Select Campaign:** A list of the exported campaigns from the source application (Lime) will be displayed so that the user can choose one of them.
- **Select Comment:** A list will be displayed with the comments included in the selected campaign, so that the user can see the details of one of them.
- **Edit Comment:** The details of a comment will be displayed, so that the user can consult and modify them.

It is left out of the project the creation of users and definition of permissions that can be done from the administration console provided by Django.

### Implementation

Now, I have implemented the identified features.

Previously to implement the features we must implement the authentication management since all the features will validate the user is logged before showing his information.

## Authentication

The first step is link the login and logout views from `django.contrib.auth.views` in the project urls file, *tfmsurveysapp/urls.py*:

```
from django.contrib import admin
from django.contrib.auth import views
urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/login/', views.LoginView.as_view(), name='login'),
    path('accounts/logout/', views.LogoutView.as_view(), name='logout'),
]
```

Then we might create the login form template in *registration/login.html*, as expected by the Django login view.

We must create the folder *tfmsurveysapp/templates* and register this folder as default templates folder in *tfmsurveys/settings.py*

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        ...
    },
]
```

Following we will create the login form in *templates/registration/login.html*

It must include some minimum elements, for example: specific form *action*, *form.username* and *form.password* input texts and hidden *next* field.

```
<form method="post" action="{% url 'login' %}">
    <table align="center" border="0" cellpadding="10"
cellspacing="0" bgcolor="#831453" style="border:solid thin black;">
        <tr>
            <td class="TextoLogin">{{ form.username.label_tag }}</td>
            <td>{{ form.username }}</td>
        </tr>
        <tr>
            <td class="TextoLogin" >{{ form.password.label_tag
}}</td>
            <td>{{ form.password }}</td>
        </tr>
        <tr>
            <td><input type="hidden" name="next" value="{{ next
}}"/></td>
        </tr>
        <tr>
            <td colspan="4" align="center">
```

```
        <input type="submit" value="login" />
    </td>
</tr>
</table>
</form>
```

We also might add in all the views a decorator to force Django redirect to the login form in case the user wasn't logged in. For example:

```
@login_required()
def campaigns_list(request):
    ...
```

And finally we must define in the file *settings.py* the default redirection file after the login or logout actions. In this case we will redirect to the root directory.

```
LOGIN_REDIRECT_URL = '/'
LOGOUT_REDIRECT_URL = '/'
```

I will start implementing the different steps that constitute each scenario and the application code to make it show the expected behaviour.

### Feature: List Campaigns

---

This is the first screen of the app and it will show a list of the campaigns exported in the Lime app.

This screen will use the *Campaign* **model** defined in the *tfmsurveysapp/models.py* file as we show in the database creation chapter.

We will create the *campaigns\_list* **view** in the configuration file *tfmsurveys/views.py*. We will create the view using the function-based strategy.

```
@login_required()
def campaigns_list(request):
    campaigns_list=Campaign.objects.all()
    context = {'campaigns_list': campaigns_list}
    return render(request, 'tfmsurveysapp/campaigns_list.html', context)
```

Inside our view we will obtain the information it must be shown in it. This is made using the Campaign model, obtaining all his objects and putting the resulting list in the context.

We also must define a template that is a html page indicating the format of the view.

The view has to return the *request* object, the name of the *template* and the *context* with information.

In the *url.py* configuration file we will define the pattern of the **url** used in the web browser to access the view.

```
path('', views.campaigns_list, name='campaigns_list')
```

In this case this will be the default page (""), it will use the *campaigns\_list* view and will also have the name *campaigns\_list* when it has to be called from other views.

And finally, we must create the **template** with the html format of the page.

Templates will be in the *tfmsurveysapp/templates/tfmsurveysapp*.

The template will contain a header and a footer common for all the templates of the project.

```
{% extends "tfmsurveysapp/base.html" %}
{% block content %}
{% load static %}
...
template's body
...
{% endblock %}
```

The main part of the template will be a table and a loop which creates a new row for every campaign.

```
{% for campaign in campaigns_list %}
    <tr>
        <td align="center">
            <a href="{% url 'tfmsurveysapp:comments_list'
campaign.cod_campania_lime %}">
                {{ campaign.cod_campania_lime }}
            </a>
        </td>
        <td>
            <a href="{% url 'tfmsurveysapp:comments_list'
campaign.cod_campania_lime %}">
```

```

campaign.cod_campania_lime %}">
                {{ campaign.name }}
            </a>
        </td>
        <td align="center">{{ campaign.type_campaign_id }}</td>
        <td>{{ campaign.type_campaign.name }}</td>
    ...
</tr>
{% empty %}<div align="center">No existeixen campanyes que
acompleixin aquestes condicions.</div>
{% endfor %}

```

The loop uses the variable stored in the context in the view and iterates through it.

Information is obtained from the object iterated that is based on the model Campaign.

We have defined a link in the code and name columns to the next view, comments\_list. To define the link, we use the url command, the name of the view in url.py and the id of the campaign. Django creates the absolute url using this information.

Note: This code is a simplification of the original code.

### Feature List Survey comments

---

This is the second screen of the app where the user can show the comments associated with a campaign.

This screen will use the *Comment* **model** and some information from his parent models *Survey* and *Campaign*.

In this case we will create the **view** using the class *CommentsList* and the generic list class (ListView) supplied by Django.

```

class CommentsList(ListView):
    model = Comment
    context_object_name = 'comments_list'
    template_name = 'tfmsurveysapp/comments_list.html'

    def get_queryset(self):
        return
Comment.objects.filter(survey__campaign__cod_campania_lime=self.kwargs['c
od_campania_lime'])

```

As in the previous feature we will define the model used for data, the name of the context variable and the html template.

To obtain the information we must create the *get\_queryset* function. In this function we filter the data from the Comment model using the *cod\_campania\_lime* passed in the url and obtained from the *kwargs* object.

The configuration of the **url** is similar to the previous feature, but we must indicate the view it's the *CommentList* class.

```
path('campaigns/<int:cod_campania_lime>',
CommentsList.as_view(), name='comments_list')
```

The contents of the **template** *comments\_list* is similar to the previous one, but with some specific details.

We iterate through the object *comments\_list*, that has been created in the view and we show the attributes of the comment object.

```
{% for comment in comments_list %}
<tr>
    <td align="center">{{ comment.survey.sid_lime }} {{
comment.id }}</td>
    <td align="left">{{ comment.survey.name }}</td>
    <td align="center">{{ comment.block_type }}</td>
    ...
</tr>
```

Some information isn't in the comment object but in some of the objects linked to it. For example: *comment* has a *survey* attribute and *survey* has a *name* attribute.

The most complex case might be obtaining the information from the campaign in which the comment is included. This information will be shown outside of the loop and for this reason we must obtain the first element of the list. For example:

```
{{ comments_list.first.survey.campaign.cod_campania_lime }}
```

## Feature Edit Comment

This feature will create a form where the user can show the details of a comment and it allows editing of some of the fields: '*Tipus d'incidencia*', '*Tipus de solucio*' and '*Comentari proposat*'.

This screen will also use the *Comment* **model** and some information from the model *Survey* and *Campaign*.

We will create the **view** using the new class *CommentDetail* and the generic editing class (*UpdateView*) supplied by Django.

```
class CommentDetail(UpdateView):
    model = Comment
    form_class = CommentForm
    template_name = 'tfmsurveysapp/comment_detail.html'
    # success_url = reverse_lazy('tfmsurveysapp:comments_list',
    kwargs={'cod_campania_lime': '170'})

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['issuetypes_list'] = IssueType.objects.all()
        context['solutiontypes_list'] = SolutionType.objects.all()
        return context

    def get_success_url(self):
        return reverse('tfmsurveysapp:comments_list',
        kwargs={'cod_campania_lime': self.kwargs['cod_campania_lime']})
```

Similar to previous features in this class we will define the model that contains the information, the name of the context variable and the html template.

But we have also to override the *get\_context\_data* function, to obtain from the *IssueType* model the list of types of issue and from the *SolutionType* model the list of types of solution. And we save the resulting list in the context.

The *UpdateView* class allows defining the *success\_url* parameter with the destination after validate and save the information contained in the form. But in this case, we need to pass to the destination address the code of the campaign and the *success\_url* doesn't allow include a variable value. To solve this problem, we must override the *get\_success\_url* function and obtain the address using the reverse function.

The next step is to create the class **form** in the *form.url* configuration file with name *CommentForm*. This class will be based on the *ModelForm* standard Django class.

```
class CommentForm(ModelForm):
    issue_type = forms.ModelChoiceField(queryset=IssueType.objects,
```

```

        required=False,
        widget=forms.Select(attrs={'class':'CampoComentario'}))
    solution_type = forms.ModelChoiceField(queryset=SolutionType.objects,
        required=False,
        widget=forms.Select(attrs={'class':'CampoComentario'}))
    new_value = forms.CharField(required=False,
        widget=forms.Textarea(attrs={'cols':'80','rows':'10',
        'class':'CampoComentario'}))

    class Meta:
        model = Comment
        fields = ('issue_type', 'solution_type', 'new_value')
        exclude = ()

```

In this class we will define the *model* attribute origin of the information, in this case the *Comment* model, and what are the editable fields.

In order to customize the fields, we must define everyone. We will indicate what *widget* will be used to show them if they are required and his class. In the case of the select fields we will define the *queryset* attribute of the *ModelChoiceField* to indicate how to obtain the information.

And finally, we must create the **template** with the html presentation of the form, that is *comment\_detail.html*.

```

<table width="100%" cellpadding="5" >
    <tr class="TituloPagina">
        <td class="EtiquetaTitulo">Campanya:</td>
        <td>{{ comment.survey.campaign.cod_campanya_lime }}</td>
        <td>{{ comment.survey.campaign.name }}</td>
        ...
    </tr>
</table>
...
<form method="post" enctype="multipart/form-data" >
    {% csrf_token %}
    ...
    <table align="center" width="50%" border="0" cellpadding="0"
    cellspacing="10" class="TablaComentario">
        <tr>
            <td class="EtiquetaComentario" width="20%">
                <label for="id_block_type">Tipus pregunta:</label>
            </td>
            <td width="15%">
                <div class="CampoComentarioInactivo">{{
comment.block_type }}</div>
            </td>
            ...
        </tr>
    ...
    <tr valign="top">

```



```

        <td class="EtiquetaComentario">Tipus incidència:</td>
        <td colspan="5">
            {{ form.issue_type }}
        </td>
    </tr>

    <tr valign="top">
        <td class="EtiquetaComentario">Proposta solució:</td>
        <td colspan="5">
            {{ form.solution_type }}
        </td>
    </tr>
    <tr valign="top">
        <td class="EtiquetaComentario">Comentari proposat:</td>
        <td colspan="5">
            {{ form.new_value }}
        </td>
    </tr>
    ...
    <tr>
        <td colspan="6" align="center">
            <input type="submit" value="Desar" />
        </td>
    </tr>
</table>
</form>

```

This html template contains different types of elements or fields.

In the first place it contains fields with information from his parent models, as the code or the name of the campaign. This information is placed outside of the form element.

It also has a *form* html element without action because Django manages the action. The *csrf\_token* directive controls the security in the use of the form.

Inside the form exists some fields that aren't editable. For this reason, they are shown from the comment model.

And the most important part is the editable fields that are obtained from the *form* object. There is no need to define the type of input and other properties because we have defined them in the forms.py file.

And lastly, it's necessary to have an input submit button.

## Results / Conclusions

A new version of the user interface has been developed that covers the basic functionalities required by the user.

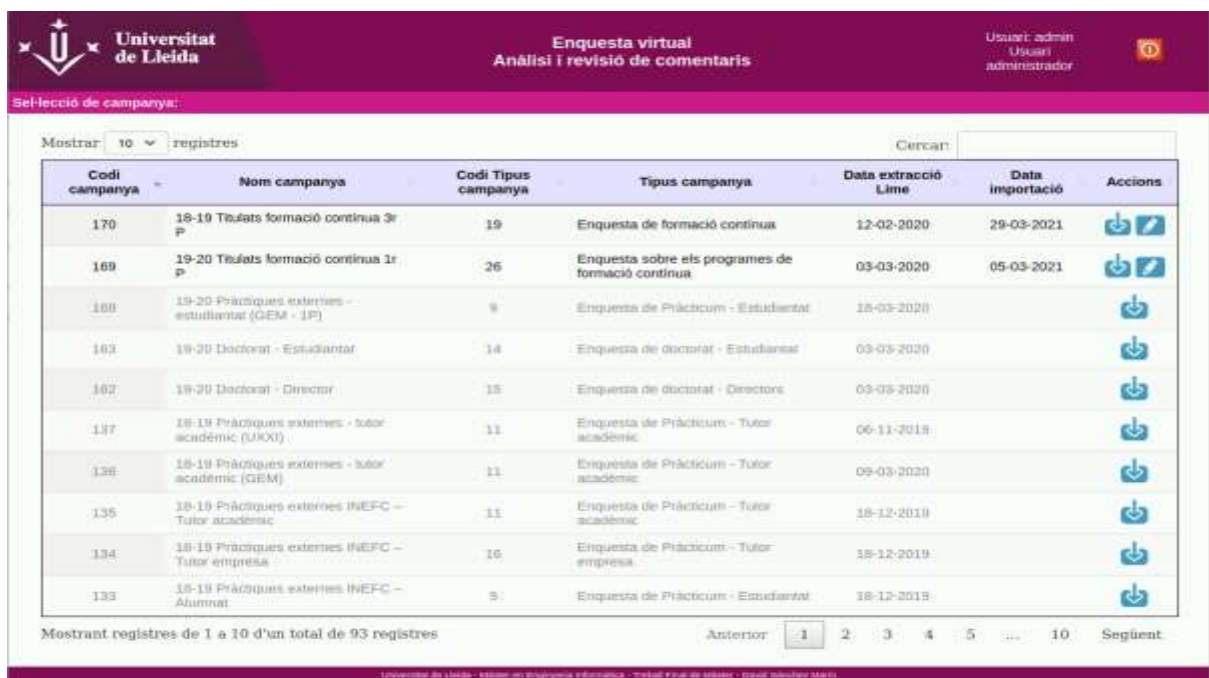
If you attempt to access any of the screens without prior authentication, the application redirects to the user validation screen:



Image 9 Login screen

After validation, the user can consult the list of campaigns exported from Lime. In this iteration are still sample data manually loaded through the database.

The campaigns with their polls and comments imported from Lime are shown in bold and have the button to view the comments. The rest of the campaigns are shown in gray. All campaigns have the button to import surveys and comments, although this functionality will be implemented in the next iteration.















Codi campanya	Nom campanya	Codi Tipus campanya	Tipus campanya	Data extracció Lime	Data importació	Accions
170	18-19 Títolats formació continua 3r p	19	Enquesta de formació continua	12-02-2020	29-03-2021	 
169	19-20 Títolats formació continua 1r p	26	Enquesta sobre els programes de formació continua	03-03-2020	05-03-2021	 
168	19-20 Pràctiques externes - estudiantat (GEM - 1P)	9	Enquesta de Pràcticum - Estudiantat	18-03-2020		
163	19-20 Doctorat - Estudiantat	14	Enquesta de doctorat - Estudiantat	03-03-2020		
162	19-20 Doctorat - Director	15	Enquesta de doctorat - Directors	03-03-2020		
137	18-19 Pràctiques externes - tutor acadèmic (UXXI)	11	Enquesta de Pràcticum - Tutor acadèmic	06-11-2019		
136	18-19 Pràctiques externes - tutor acadèmic (GEM)	11	Enquesta de Pràcticum - Tutor acadèmic	09-03-2020		
135	18-19 Pràctiques externes INEFC - Tutor acadèmic	11	Enquesta de Pràcticum - Tutor acadèmic	18-12-2019		
134	18-19 Pràctiques externes INEFC - Tutor empresa	10	Enquesta de Pràcticum - Tutor empresa	18-12-2019		
133	18-19 Pràctiques externes INEFC - Alumnat	9	Enquesta de Pràcticum - Estudiantat	18-12-2019		

Image 10 List of campaigns screen

When a campaign is selected, the list of surveys and comments included in it is displayed. The user can access the Edit Comment screen using the button included for each row.

sid	Descripció	Tipus preg.	Codi pregunta	Pregunta	sid	Comentari original	Tipus incidència	Solució	Comentari proposat	Accions
186122 270479603	Enquesta als titulats/es en MÀSTER DUAL EN DIRECCIÓ D'OPERACIONS I DISTRIBUCIÓ - 044M	P	PR020	Punts forts / Àrees de millora	3	Falta organització en l'estructura de les matèries. Hi ha molts continguts i coneixements, però sense aprofundir.	Comentari de professor	Eliminar professor	Falta organització en l'estructura de les matèries. Hi ha molts continguts i coneixements, però sense aprofundir.	
186122 270479604	Enquesta als titulats/es en MÀSTER DUAL EN DIRECCIÓ D'OPERACIONS I DISTRIBUCIÓ - 044M	P	PR020	Punts forts / Àrees de millora	4	Punts a millorar: Organització de les assignatures, Contingut de les assignatures, Organització i lideratge per part de la direcció. No es tracten tots els temes plantejats als projectes del màster.	Comentari d'assignatura	Copiar al bloc d'assignatura	xxxx	
255385 270482903	Enquesta als titulats/es en DISENY I GESTIÓ BIM TREBALL FINAL DE MÀSTER - 319C	P	PR020	Punts forts / Àrees de millora	3	Igualar el temps de practiques i de teoria. Incrementar el temps del mòdul CA.	Comentari problemàtic		Igualar el temps de practiques i de teoria. Incrementar el temps del mòdul CA.	

Image 11 Lista of surveys and comments screen

Finally, you access the screen to view and edit the comments data. It is possible to modify the fields of Type of incidence, Solution proposal and Comment proposal.

Tipus pregunta: P Codi pregunta: PR020 Titl.: 3

Pregunta: Punts forts / Àrees de millora

Comentari original: Falta organització en l'estructura de les matèries. Hi ha molts continguts i coneixements, però sense aprofundir.

Tipus incidència: Comentari d'assignatura

Preposta solució: Copiar al bloc d'assignatura

Comentari proposat: Falta organització en l'estructura de les matèries. Hi ha molts continguts i coneixements, però sense aprofundir.

Desar

Image 12 Edit comment screen

After modifying a comment, go back to the campaign comment list.

We have taken advantage of the advantages offered by Django for the implementation of web applications and have explored different alternatives to solve the problems and to know a little more about the Framework.

## Iteration 7. Import surveys

### Requirements / Specification

In this iteration, we will develop the functionality of importing the surveys and comments of a campaign from Lime application databases.

The first step is to import the type of campaigns and the campaigns. This would be done when the user accesses the app.

Following this, the user will execute the process of import campaigns from the list of campaigns using the import button. After importation, the list of campaigns will contain the date of the importation and the open comment button will be shown.

It's possible to import a campaign more than once and the previous surveys will be deleted. The user would be warned about this before executing the operation.

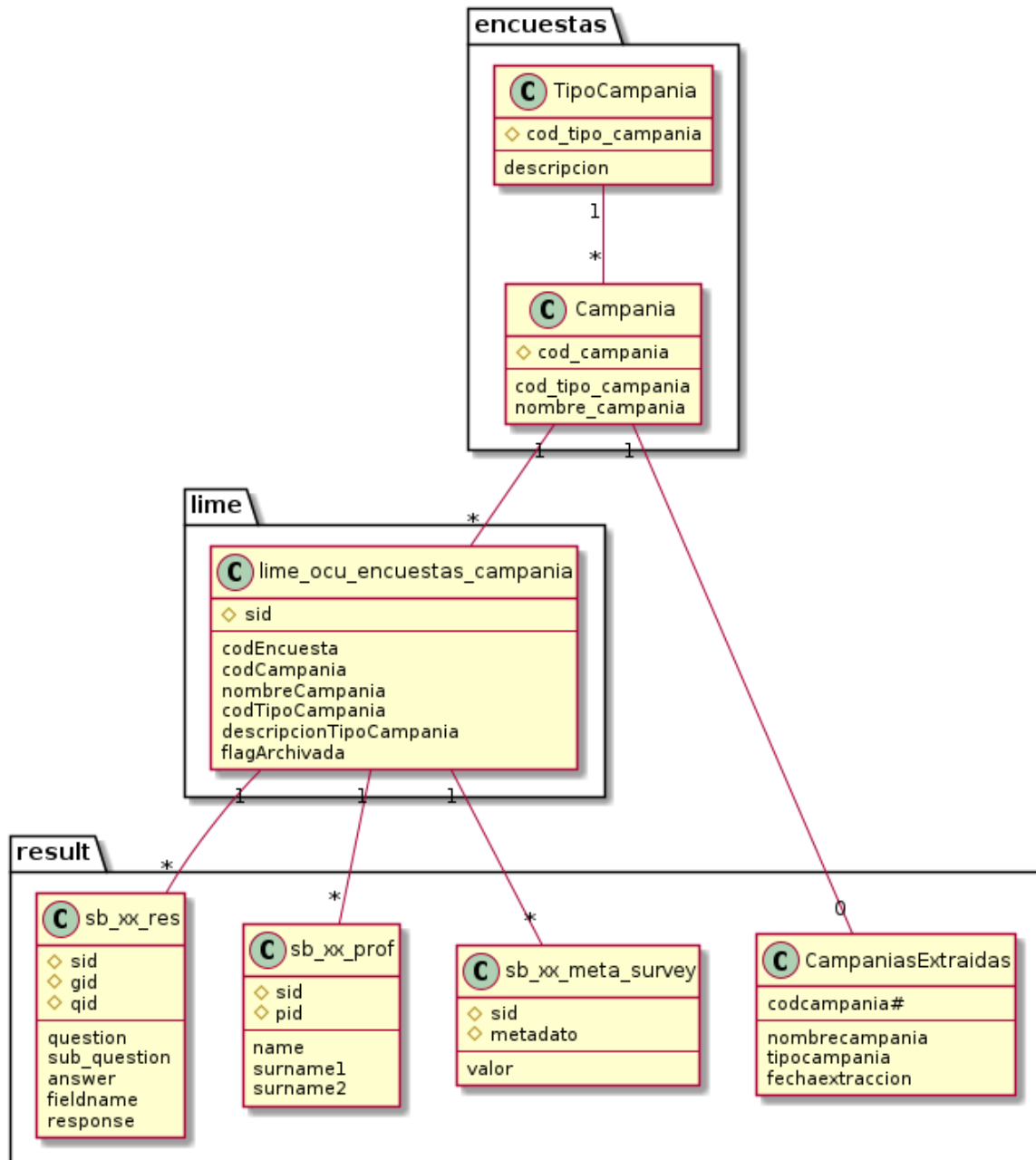
Codi campanya	Nom campanya	Codi Tipus campanya	Tipus campanya	Data extracció Lime	Data importació	Accions
170	18-19 Treballs formació contínua 3r P	19	Enquesta de formació contínua	12-02-2020	29-03-2021	
169	19-20 Treballs formació contínua 1r P	26	Enquesta sobre els programes de formació contínua	03-03-2020	05-03-2021	
168	19-20 Pràctiques externes - estudiantat (GEM - IP)	9	Enquesta de Pràcticum - Estudiantat	18-03-2020		
163	18-20 Doctorat - Estudiantat	14	Enquesta de doctorat - Estudiantat	03-03-2020		
162	18-20 Doctorat - Director	15	Enquesta de doctorat - Directors	03-03-2020		
137	18-19 Pràctiques externes - tutor acadèmic (JXXI)	11	Enquesta de Pràcticum - Tutor acadèmic	06-11-2019		
136	18-19 Pràctiques externes - tutor acadèmic (GEM)	11	Enquesta de Pràcticum - Tutor acadèmic	09-03-2020		
135	18-19 Pràctiques externes INEFC - Tutor acadèmic	11	Enquesta de Pràcticum - Tutor acadèmic	18-12-2019		
134	18-19 Pràctiques externes INEFC - Tutor empresa	10	Enquesta de Pràcticum - Tutor empresa	18-12-2019		
133	18-19 Pràctiques externes INEFC - Alumnat	9	Enquesta de Pràcticum - Estudiantat	18-12-2019		

Image 13 Lista of campaigns screen

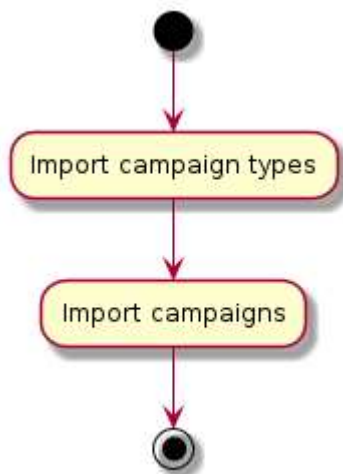
## Design

The first challenge we must solve is accessing the original databases from the Lime app. It has 3 schemas with some tables interrelated.

The following graph shows the tables and main fields used in the importation process:



Schema 18 Partial Lime database

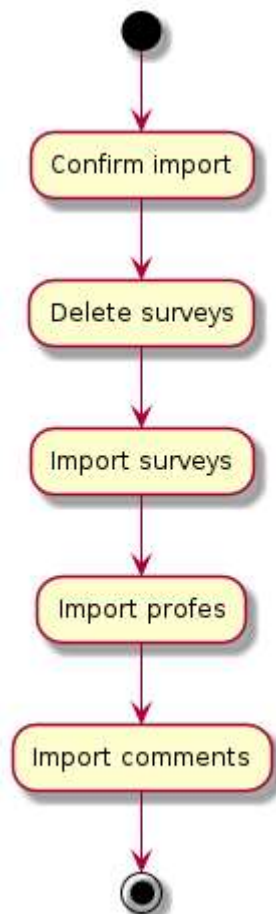


Schema 19 Activity graph import campaigns

The process of importation of types of campaigns and campaigns will be executed every time the user accesses the application.

It might copy information from tables '*TipoCampania*' and '*Campania*' in '*Encuestas*' database to '*CampaignType*' and '*Campaign*' models in '*tfmsurvey*' database.

The process would consist in creating new records, updating modified records and delete records that don't exist anymore.



Schema 20 Activity graph import surveys and comments

The process of importing surveys will be started by the user clicking the 'import' button. The steps would be:

- **Confirm import:** The app would detect if the campaign has been previously imported and ask the user for his confirmation.
- **Delete surveys:** All the previous information from the campaign would be deleted. This includes surveys, professors and comments.
- **Import surveys:** The general information of the survey would be copied from the '*sb\_xx\_meta\_survey*' table in '*result*' schema to the '*Survey*' model in the '*tfmsurvey*' database.
- **Import professors:** In surveys of type '*assignatura-professor*' the professor information would be copied from the '*sb\_xx\_meta\_survey*' table to the '*Professor*' model in '*tfmsurvey*'.
- **Import comments:** The comments entered by the user would be copied from the '*sb\_xx\_res*' table in the '*result*' schema to the '*Comment*' model in the '*tfmsurvey*' database.

## Implementation

The implementation of this feature includes some steps that will be explained in detail in this chapter.

It will be explained the new concepts of Django that had been used:

- Multiple database management
- Legacy database access
- Multifield keys issue
- Virtual model
- Use of RedirectView
- Batch of bulk operations
- Raw queries
- 

### Multiple databases management

This project must interact with 4 different databases. Three of them are legacy databases from the original application and will allow read-only access (*encuestas*, *lime* and *uxxienc\_resul*). The other one is the application database and it will allow read-write access (*default*).

The first step is to tell Django the **settings** to connect with the database servers. This is done in the *DATABASE* block of the *settings.py* configuration file.

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'tfm_surveys',
        'USER': 'tfm',
        'PASSWORD': '****',
        'HOST': 'localhost',
        'PORT': '3306'
    },
    'encuestas': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'encuestas',
        'USER': 'tfm',
        'PASSWORD': '****',
        'HOST': 'localhost',
        'PORT': '3306'
    },
    'lime': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'lime',
        'USER': 'tfm',
        'PASSWORD': '****',
        'HOST': 'localhost',
        'PORT': '3306'
    },
    'uxxienc_resul': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'uxxienc_resul',
        'USER': 'tfm',
        'PASSWORD': '****',
        'HOST': 'localhost',
    }
```



```
}
    'PORT': '3306'
}
```

Only the default database will be managed from Django, because the other databases were previously created and we haven't the capacity to edit them.

To use multiple databases is necessary to set up a database **routing scheme**. This routing scheme directs models to their original database. And models that aren't specified will go to the default database schema.

A database routing is a class defined in the *router.py* configuration file located in the *tfmsurveysapp*. This class provides four methods:

- *db\_for\_read*: Suggest the database that should be used for read operations.
- *db\_for\_write*: Suggest the database that should be used for write operations.
- *allow\_relation*: Returns if a relation between two models must be allowed.
- *allow\_migrate*: Indicate if the migration operation is allowed.

In our case the name of the apps is the same as the name of databases, relation will be allowed only between models in the same app and migration is allowed if app and database have the same name.

This is part of the router code:

```
class EncuestasRouter:

    route_app_labels = {'encuestas', 'lime', 'uxxienc_resul'}

    def db_for_read(self, model, **hints):
        if model._meta.app_label in self.route_app_labels:
            return model._meta.app_label
        return None

    def db_for_write(self, model, **hints):
        if model._meta.app_label in self.route_app_labels:
            return model._meta.app_label
        return None

    def allow_relation(self, obj1, obj2, **hints):
        if (obj1._meta.app_label == obj2._meta.app_label):
            return True
        else:
            return None

    def allow_migrate(self, db, app_label, model_name=None, **hints):
        if app_label in self.route_app_labels:
            return (db == app_label)
        return False
```

## Legacy databases access

The second step is to create the **models** to query the information from the tables.

In order to make it easy to Django the connection between tables and models, three more applications are created. Every database will have its own application.

This action will be done with the commands:

```
python3 manage.py startapp encuestas
python3 manage.py startapp lime
python3 manage.py startapp uxxienc_resul
```

These new apps will only have the function of containing the model definition of the corresponding database.

To speed the creation of models from an existing database Django comes with a utility called *inspectdb*. That utility creates models by introspecting the existing database and it has been used to create a base model definition file for every application (*models\_inspectdb.py*). The commands used to create the initial model file are:

```
python3 manage.py inspectdb --database encuestas > models_inspectdb.py
python3 manage.py inspectdb --database lime > models_inspectdb.py
python3 manage.py inspectdb --database uxxi_enc > models_inspectdb.py
```

Then, the needed models had been copied and customized to the final *models.py* file in every application. To avoid Django modifying the original table's structure, the *managed = False* metadata field has been defined.

## Encuestas / models.py

```
class TipoCampania(models.Model):
...
    class Meta:
        managed = False
        db_table = 'TIPO_CAMPANIA'

class Campania(models.Model):
...
    class Meta:
        managed = False
        db_table = 'CAMPANIA'

class Encuesta(models.Model):
...
    class Meta:
        managed = False
        db_table = 'ENCUESTA'
```

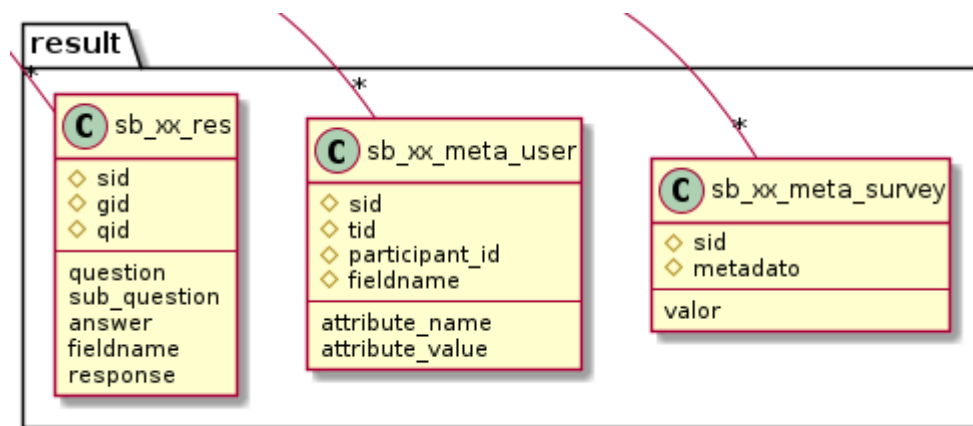
## Lime / models.py

```
class LimeOcuEncuestasCampania(models.Model):  
...  
    class Meta:  
        managed = False  
        db_table = 'lime_ocu_encuestas_campania'
```

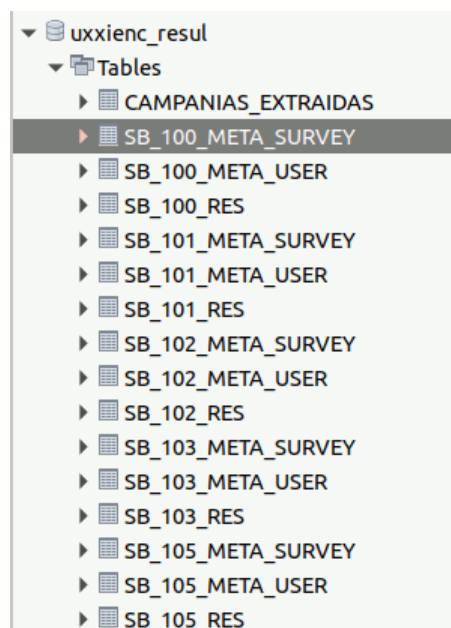
## Uxxienc\_resul / models.py

Models from this database are different from the previous ones.

The CampaniasExtraidas model is similar to the previously created models, but every campaign has 3 tables with the same structure between campaigns and with the code of the campaign in his name. In this schema 'xx' is the campaign code).



Schema 21 Uxxienc\_resul database models



Schema 22 Examples of Uxxienc\_resul tables

Only one model has been created for every type of 'SB' table where it is not possible to define the table name and it will force access to information using **dynamic queries**.

Another particular characteristic of this table is that they use **multifield keys**. The primary key is the composition of two or more fields. Django doesn't support this functionality, but it's possible to have an approximation defining as *primary\_key = True* the first of the key fields and adding to the metadata the *unique\_together* property with the relation of key fields.

```
class CampaniasExtraidas(models.Model):
...
    class Meta:
        managed = False
        db_table = 'CAMPANIAS_EXTRAIIDAS'

class SBMetaSurvey(models.Model):
    sid = models.IntegerField(blank=True, primary_key=True)
    metadato = models.CharField(max_length=20, blank=True, null=True)
...

    class Meta:
        unique_together = (('sid', 'metadato',))
        managed = False
#        db_table = 'SB_100_META_SURVEY'

class SBMetaUser(models.Model):
    sid = models.IntegerField(blank=True, primary_key=True)
    tid = models.IntegerField(blank=True, null=True)
    fieldname = models.CharField(max_length=12, blank=True, null=True)
...

    class Meta:
        unique_together = (('sid', 'tid', 'fieldname',))
        managed = False
#db_table = 'SB_100_META_USER'

class SBRes(models.Model):
    sid = models.IntegerField(blank=True, primary_key=True)
    tid = models.IntegerField(blank=True, null=True)
    gid = models.IntegerField(blank=True, null=True)
...

    class Meta:
        unique_together = (('sid', 'tid', 'qid',))
        managed = False
#db_table = 'SB_100_RES'

class SBProf(models.Model):
    sid = models.IntegerField(null=False, primary_key=True)
    pid = models.CharField(max_length=2, null=False)
...

    class Meta:
        unique_together = (('sid', 'pid',))
        managed = False
```

Professor information is a special case. This information doesn't come from a specific table but it is obtained from the *SB\_XX\_META\_SURVEY* table through a query and saved in a virtual model with a convenient structure.

## Campaign types and Campaigns importation

The importation of campaign types and campaigns is done in the *import\_campaign\_types* and *import\_campaigns* functions, located in the *views.py* file.

These functions are called in the *campaigns\_list* view, every time the view is opened.

Both functions have the same structure.

Original records are read using the *TipoCampania* model in the *Encuestas* app or the *CampaniasExtraidas* model in the *uxxienc\_resul* app.

Final information is stored in *CampaignType* objects in the *tfmsurveysapp* or in *Campaign* objects in the *tfmsurveysapp*.

Using a loop, the function detects if records exist and if they are modified, in order to create or update them.

Using another loop, the function detects records that no longer exist and delete them.

This is the code of the function *import\_campaign\_types* function:

```
def import_campaign_types():
    tipocampanias_lime = TipoCampania.objects.all()

    # New and update campaign types
    for tipocampania_lime in tipocampanias_lime:
        try:
            campaigntype_tfm = CampaignType.objects.get(
                cod_tipo_campania_lime =
                    tipocampania_lime.cod_tipo_campania)
            if (campaigntype_tfm.name != tipocampania_lime.descripcion):
                campaigntype_tfm.name = tipocampania_lime.descripcion
                campaigntype_tfm.save()

        except CampaignType.DoesNotExist:
            new_campaigntype = CampaignType(
                cod_tipo_campania_lime =
                    tipocampania_lime.cod_tipo_campania,
                name = tipocampania_lime.descripcion)
            new_campaigntype.save()

    # Delete campaign types
    campaigntypes_tfm = CampaignType.objects.all()
    for campaigntype_tfm in campaigntypes_tfm:
        try:
            tipocampania_lime = TipoCampania.objects.get(
                cod_tipo_campania = campaigntype_tfm.cod_tipo_campania_lime)
        except TipoCampania.DoesNotExist:
            campaigntype_tfm.delete()
```

Campaign information has an additional issue. When creating new campaigns, the program

obtains the id of the campaign type but *Campaign* model needs a *CampaignType* object. It's necessary to get the corresponding object from the *CampaignType* model.

```
def import_campaigns():
    campanias_lime = CampaniasExtraidas.objects.all()

    # New and update campaign types
    for campania_lime in campanias_lime:
        ...
        campaign_type_tfm = CampaignType.objects.get(
            name=campania_lime.tipocampania)
        new_campaign = Campaign(
        ...
            type_campaign=campaign_type_tfm
        )
        new_campaign.save()
    ...
```

## Import surveys

The importation of surveys and all the associated information is done from the *ImportCampaign* view, that is a **RedirectView**. This type of Django views automates the execution of operations and redirection to a preexisting url. The main properties of this type of view are:

- *url*: The fix URL to redirect to.
- *pattern\_name*: The name of the URL pattern to redirect to, defined in *url.py*. It will use the same arguments passed in to the view.

In this case it isn't possible to use these properties because it's necessary to redirect to the url without an argument and Django automatically adds the received arguments in the original call.

It has been necessary to implement the *get\_redirect\_url* method to create a customized url without arguments. Also, the importing function has been included in this method after obtaining the *cod\_campania\_lime* argument.

This is the code of the *ImportCampaign* view:

```
class ImportCampaign(RedirectView):
    query_string = False
    permanent = False

    def get_redirect_url(self, *args, **kwargs):
        cod_campania_lime = kwargs['cod_campania_lime']
        self.delete_surveys(cod_campania_lime)
        self.import_surveys(cod_campania_lime)
        self.import_profes(cod_campania_lime)
        self.import_comments(cod_campania_lime)
        self.update_import_date(cod_campania_lime)
```

```
return reverse('tfmsurveysapp:campaigns_list')
```

The functions created to import surveys and his associated information are:

- delete\_surveys
- import\_surveys
- import\_professors
- import\_comments
- update\_import

### Delete surveys function

Obtain the previous surveys filtering by *cod\_campania\_lime* and erasing these records using the *delete* method. Information in models associated with *Survey* will be also deleted because the on delete cascade option is defined in the *Survey* model.

```
def delete_surveys(self, cod_campania_lime):  
    surveys = Survey.objects.filter(  
        campaign__cod_campania_lime=cod_campania_lime).delete()  
    return True
```

### Import Surveys function

Original information is read mainly from *Encuesta* model in the *Encuestas* app, but the campaign object is read from *Campaign* model in *tfmsurveysapp* and the *sid* attribute is read from the *LimeOcuEncuestasCampania* in *uxxienc\_resul* app. Final information is saved in the *Survey* model in *tfmsurveysapp* using the *save* method.

This implementation has a problem of **efficiency**. For example, importing 500 registers takes 67 seconds. And all the other tables associated with surveys could have the same problem and they have a bigger cardinality.

The solution is the use of the method **bulk\_create**, where registers are stored in a list and saved to disk in a batch operation. In this case the execution time is 0,67 seconds. In other words, 100 times faster. Different batch sizes had been tried to evaluate performances, but there isn't a significant variation in the execution time.

```
def import_surveys(self, cod_campania_lime):  
    campaign = Campaign.objects.get(cod_campania_lime=cod_campania_lime)  
    encuestas = Encuesta.objects.filter(campania_id=cod_campania_lime)  
    surveys_list = []  
    ...  
    for encuesta in encuestas:  
        try:  
            limeencuesta = LimeOcuEncuestasCampania.objects.get(  
                codencuesta=encuesta.cod_encuesta)  
            ...
```

```

        survey = Survey(
            campaign = campaign,
            cod_encuesta_lime = encuesta.cod_encuesta,
            sid_lime = limeencuesta.sid,
            name = encuesta.titulo
        )
        surveys_list.append(survey)
        #survey.save()
    ...
    Survey.objects.bulk_create(surveys_list)
    ...

```

## Import Professors function

In this operation there doesn't exist a table in the original database that contains the professor's information; however, a complex query is needed to obtain the information. This issue could be resolved by performing **raw queries**.

In order to store the information obtained with this query a model called *Professor* had been created, with exactly the same fields obtained in the query. This data will be saved in the *Professor* model in the *tfmsurveysapp*.

This is the main code of *import\_professor* function:

```

def import_profes(self, cod_campania_lime):

    query = "SELECT prof.sid, " \
            "LPAD(prof.num_profe, 2, '0') pid, " \
            "MAX(IF(metadata = 'APELLIDO1PROFE', valor, null)) surname1, " \
            "\
            "MAX(IF(metadata = 'APELLIDO2PROFE', valor, null)) surname2, " \
            "\
            "MAX(IF(metadata = 'NOMBREPROFE', valor, null)) name " \
            "FROM " \
    ...
    profes = SBProf.objects.raw(query, [cod_campania_lime])
    profes_list = []
    for profe in profes:
        ...
        survey = Survey.objects.get(sid_lime = profe.sid)
        ...
        tfmprofe = Professor(
            sid_lime = profe.sid,
            pid_lime = profe.pid,
            name = profe.name,
            surname1 = profe.surname1,
            surname2 = profe.surname2,
            survey = survey
        )
        profes_list.append(tfmprofe)
    ...

    return True

```



## Import Comments function

**Raw queries** had also been used in this operation, because the name of the original table is different for each campaign unlike the structure of all the tables is the same. The SBRES model had been created to save the information from these tables. Comment is the destination model in *tfmsurveysapp*.

Another special situation in this table is that only some types of campaigns and some types of questions include professor information, data about the professor must be obtained from the *Professor* model and the name of the fields with name and surname of the professor include a variable number.

```
def import_comments(self, cod_campania_lime):

    campaign = Campaign.objects.get(cod_campania_lime = cod_campania_lime)
    if ('assignatura' in campaign.type_campaign.name):
        survey_type = 1      # Assignatura-professor
    else:
        survey_type = 2      # Altres enquestes

    comments_list = []
    query = "SELECT r.sid, r.tid, r.gid, r.type, r.parent_qid, r.qid, " \
           "r.sqid, r.ssqid, r.question, r.sub_question, " \
           "r.sub_sub_question, r.answer, r.fieldname, r.response, " \
           "r.token, q.title question_id " \
           "FROM uxxienc_resul.SB_%s_RES r INNER JOIN lime.lime_questions q" \
           " ON r.qid = q.qid " \
           "WHERE r.type = 'T' AND q.language = 'ca' AND r.response > ''"
    comments = SBRes.objects.raw(query, [cod_campania_lime])

    for comment in comments:
        ...
        survey = Survey.objects.get(sid_lime = comment.sid)
        question = comment.question
        # Obtain professor
        if survey_type == 1:
            if len(comment.question_id) > 3:
                block_type = 'P'
            else:
                block_type = 'A'
            pid = comment.question_id[3:5]
            if pid != "":
                try:
                    professor = Professor.objects.get(
                        sid_lime=comment.sid, pid_lime=pid)

                    question = self.replace_macro(question,
                                                    "NOMBREPROFE", professor.name)
                    question = self.replace_macro(question,
                                                    "APELLIDO1PROFE", professor.surname1)
                    question = self.replace_macro(question,
                                                    "APELLIDO2PROFE", professor.surname2)

                    ...
                    tfmcomment = Comment(
                        survey = survey,
                        qid_lime = comment.qid,
```

```

        tid_lime = comment.tid,
        question_id_lime = comment.question_id,
        question = question,
        block_type = block_type,
        professor = professor,
        original_value = comment.response
    )
    comments_list.append(tfmcomment)
    ...
    Comment.objects.bulk_create(comments_list)

    return True

```

## Update import date

This is the final operation after importing surveys where the campaign is marked with the date of importation. The campaign is obtained from the *Campaign* model of the *tfmsurveysapp* and the *import\_date* field updated with the current date.

```

def update_import_date(self, cod_campania_lime):
    campaign = Campaign.objects.get(cod_campania_lime=cod_campania_lime)
    campaign.import_date = date.today()
    campaign.save()

```

## Results / Conclusions

After the implementation of this iteration the application has the possibility of importing information from the original application by demand of the user.

I have explored most of the mechanisms of Django to manage information and view, for instance:

- Multiple database management
- Legacy database access
- Multifield keys issue
- Virtual model
- Use of RedirectView
- Batch of bulk operations
- Raw queries

## Iteration 8. Integration of Django and Spacy - Process comments

### Requirements / Specification

In this iteration we will integrate web development done using Django with the Artificial Intelligence functions created with Spacy. This task will force us to adapt the resulting function from Spacy and the views and templates to call the functions and show the results.

It will be necessary to create two classes that will control language detection and the classification of comments that match the issue type called 'model 1 - professor don't have participated in the subject'.

The language detection will be called when reading the comments from the import surveys functionality and it will save the detected language as an attribute of the comment.

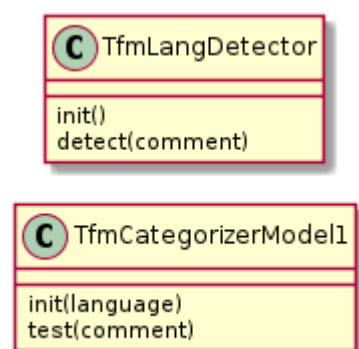
In order to know the result of the language detection the comments list page will show a summary with the number of comments in every language. Also will be possible filter by language, to validate the detection correctness and change the language value in the edit comment form.

The issue type detection will be called from the campaigns list with a new button in a similar way as done with the import campaign functionality. This button will be also in the comments list page.

The result of the issue detection could be shown as a column in the comments list and a summary in the header of this page. It will be possible to filter the detected issues and change the issue type from the edit comment page.

### Design

It would be done with a lot of little changes in the code developed until now that we will define in this part.



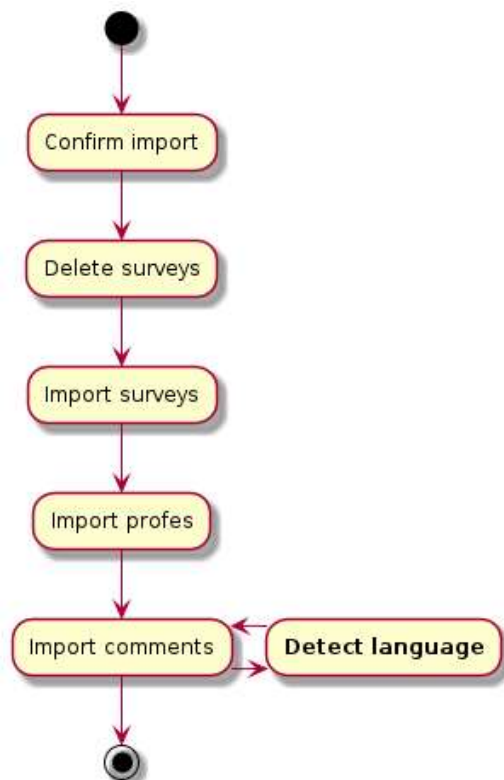
Schema 23 Spacy classes graph

Class ***TfmLangDetector*** will include two methods:

- ***init***: Create de Spacy model and prepare the LanguageDetector tool.
- ***detect***: Receive the comment to analyze and return the code of the detected language (ca, es, en).

Class ***TfmCategorizerModel1*** will include two methods:

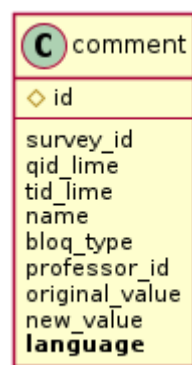
- ***init***: Load the Spacy trained model of the selected language
- ***test***: Receive a comment and calculate the probability to be included in the issue type 1.



Schema 24 Activity graph import surveys with language detection

A new function **Detect language** will be called from the *Import comments* function of the *Import surveys* process.

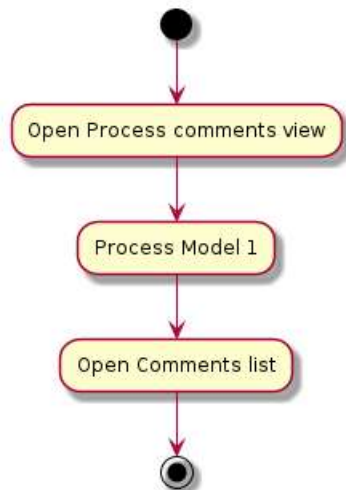
This function will obtain the language of the comment and save it in a field of the comment called **language**.



Schema 25 Model Comments modification

The **summary** of comments by language will be calculated using a Django grouping method and the **filter** will be done adding a link to in every number redirecting to the url: campaigns/<cod\_campania>/lang/<language\_code>

Code	Language	Comments	url
ca	Català	26	campaigns/123/lang/ca
es	Castellà	15	campaigns/123/lang/es
en	Anglès	6	campaigns/123/lang/en
	Total	47	campaigns/123



The detection of the issue type 1 will be done in a new **view** called **Process comments**.

This view will execute a **function** called **Process Model 1** that will loop through all the languages and all the comments and evaluate them saving the issue type if they pass the test.

And after it the program will be redirected to the '**Comments list**' view.

Schema 26 Activity graph Process model 1

The **summary** of comments by issue type will be calculated counting the comments detected and the **filter** will be done adding a link to in the number of comments redirecting to the url: campaigns/<cod\_campania>/issue\_type/1. In this occasion will be used the constant '1't, but in the future could be used a variable when more models were implemented.

Code	Language	Comments	url
1	Model 1	13	campaigns/123/issue_type/1
	Total	47	campaigns/123

## Implementation

The implementation of this iteration has consisted in the following steps:

Creation of classes TfmLangDetect and TfmCategorizerModel1

The code related with *Spacy* implementation has been placed in the folder '*tfmsurveysapp/spacy*'. The resulting models from the Spacy training are located in the folder "*tfmsurveysapp/spacy/nlp\_models*".

The final classes are relatively simple because all the work was made in his initial analysis. As explained in the design part **TfmLangDetector** has two functions:

- **init**. Load the English standard model and add the LanguageDetector class in the pipe that uses Spacy

```
def __init__(self):
```

```

...
self.nlp = spacy.load("en_core_web_sm")
self.nlp.add_pipe(LanguageDetector(), name='language_detector',
                  last=True)
...

```

- **detect:** This function receives a comment and mixes the language detected by Spacy and Pycl2, to obtain an improved prediction. In the language detection iteration was explained on how this mix is done. This is part of the code:

```

def detect(self, comment):
    # Language detected by Spacy
    lang_spacy = self.nlp(comment)._.language["language"]

    # Language detecte by Pycl2
    isReliable, textBytesFound, details =
        cld2.detect(comment.encode('utf-8', 'replace'),
                    isPlainText=True, bestEffort=True, returnVectors=False)
    lang_pycl2 = details[0][1]

    # Mix of predictions
    language = lang_spacy
    if (lang_spacy != "ca") & (lang_pycl2 == "ca"):
        ...
    return language

```

The class `TfmCategorizerModel1` has also two functions as described in the design:

- **init:** Loads the model created in the training iteration. Every language has his own model placed in a subfolder of the folder `tfmsurveysapp/models`. For example: `nlp_models/ca`, `nlp_models/es` or `nlp_models/en`.
- **test:** This function creates a *Spacy doc* object from the received comment. This object contains the *cats* object that indicates the probability that the comment matches with the issue type '1 - Professor hasn't done classes'.

This is an example of the *cats* object, where we can show the probability that the comment matches this type of issue is more than 99%:

```

{'POSITIVE': 0.9999936819076538, 'NEGATIVE':
6.376371402438963e-06}

```

This is the code of the class:

```

class TfmCategorizerModel1():

    def __init__(self, language):
        ...
        self.nlp = spacy.load(pathmodel + language)
        ...

```

```
def test(self, comment):
    doc = self.nlp(comment)
    ...
    return doc.cats
```

### Calculating language in ImportCampaign view

The calculation of the comment language had been added in the *import\_comments* function that is included in the *ImportCampaign* view.

In the beginning of the function *lang\_detector* variable is created as an instance of *TfmLangDetector*. This action executes automatically the init function of the class.

As shown in the previous iteration, comments are read using a raw query and iterating through them.

The *language* is obtained by calling the *detect* function of the *lang\_detector* object.

And the *language* information is stored in the *tfmcomment* object and the *tfmcomment* is saved to disk.

This is the part of the code added in this iteration to manage the language detection:

```
class ImportCampaign(RedirectView):
    ...
    def get_redirect_url(self, *args, **kwargs):
        ...
        self.import_comments(cod_campania_lime)
        ...

    def import_comments(self, cod_campania_lime):

        # Init language detector
        lang_detector = TfmLangDetector()
        ...
        comments = SBRes.objects.raw(query, [cod_campania_lime])
        for comment in comments:
            ...
            lang = lang_detector.detect(comment.response)
            ...
            tfmcomment = Comment(
                ...
                language = lang
            )
            comments_list.append(tfmcomment)
        ...
        Comment.objects.bulk_create(comments_list)
        ...
```

## Modification of *Comment* model

The *Comment* model had been modified to add the language field where save the language detected in the previous step. It will contain the code of the language (ca, es or en)

```
class Comment(models.Model):  
    ...  
    language = models.CharField("Idioma", max_length=2, null=True)
```

## Modification in template *comments\_list.html* to identify language

It has been added the 'Idioma' column to the table of comments:

```
...  
<table align="center" id="Lista" name="Lista" class="display">  
    <tr>  
        ...  
        <th align="center">Idioma</th>  
        ..  
    </tr>  
    {% for comment in comments_list %}  
    <tr>  
        ...  
        <td align="center">{{ comment.language }}</td>  
        ...  
    </tr>  
    ...
```

## Modification in templates to process issue type of comments

To execute the **process** of identifying **comments** of issue type 1 has been added a button in the *campaign\_list* and *comments\_list* templates. This button calls a script function that redirects the page to the *process\_comments* view with the parameter *cod\_campania\_lime*.

This is the code added to the *campaign\_list*:

```
<a onclick="processarComentaris('{% url 'tfmsurveysapp:process_comments'  
    campaign.cod_campania_lime %}')"  
    ...  
</a>
```

And this is the code added to the *comments\_list*:

```
<a onclick="processarComentaris('{% url 'tfmsurveysapp:process_comments'  
    comments_list.first.survey.campaign.cod_campania_lime %}')"  
    ...  
</a>
```



And this is the script who made the redirection:

```
function processarComentarios(url) {  
    ...  
    window.location.href = url;  
}
```

### Creation of ProcessComments view

The main part in the task of process comments is done by the *ProcessComments* view, included in the *views.py* configuration file.

This view obtains the *cod\_campania\_lime* parameter from the url, executes the *process\_model1* function and redirects the page to the *comments\_list* view using the *pattern\_name* attribute of the view.

The *process\_model1* function executes the following steps:

1. Iterate through the possible languages (ca, es, en).
2. Load the model corresponding to the language, creating a *nlp* object, that is an instance of the *TfmCategorizerModel1* class. This step calls the *init* function of the class.
3. Filter the comments by the desired language.
4. Iterate through the comments.
5. For every comment evaluate if the comment matches with the issue type. This action calls the *test* function of the *nlp* object, that is an instance of the *TfmCategorizerModel1* class.
6. If the result of the test is higher than 50% the *issue\_type* field is completed and the comment saved with the new value. Percentage can be adjusted to be more accurate in the selection of comments.

This is the code of the *ProcessModel1* view:

```
class ProcessComments(RedirectView):  
    query_string = False  
    permanent = False  
    pattern_name = "tfmsurveysapp:comments_list"  
  
    def get_redirect_url(self, *args, **kwargs):  
        cod_campania_lime = kwargs['cod_campania_lime']  
        self.process_model1(cod_campania_lime)  
  
        return super().get_redirect_url(*args, **kwargs)
```

And this is the more relevant code of the *proces\_model1* function:

```
def process_model1(self, cod_campania_lime):

    issue_type = IssueType.objects.get(id=1)
    languages = {"ca","es","en"}
    for language in languages:
        nlp = TfmCategorizerModel1(language)

        comments = Comment.objects.filter(
            survey__campaign__cod_campania_lime=
            cod_campania_lime,language=language)

        for comment in comments:
            result = nlp.test(comment.original_value)
            if (result['POSITIVE'] > 0.5):
                comment.issue_type = issue_type
                comment.save()

    return True
```

#### Definition of url to process comments

Next step consists in defining the url to access the *ProcessComments* view in the *url.py* configuration file. Example: campaigns/123/process

```
path('campaigns/<int:cod_campania_lime>/process',
      ProcessComments.as_view(),
      name='process_comments'),
```

#### Summary of languages and issue types

So as to validate the language and issue types calculation have been added both summary tables in the comments list page. Calculation is done in the *CommentList* view.

In the case of language summary values of *comments\_list* are grouped using the value method and making a count using the *annotate* method. This returns a list of language codes and number of comments.

In the case of issue type summary *comments\_list* is filtered by issue type and the count method returns the final number. This codification is possible because there is only one type of issue at this moment.

Summaries are stored in the context to be recovered in the template.

This is the part of code corresponding to summary calculation:

```
class CommentsList(ListView):
    ...

    def get_queryset(self):
        ...
        comments_list = Comment.objects.filter(
            survey__campaign__cod_campania_lime=
            self.kwargs['cod_campania_lime'])
        self.languages_summary = comments_list.values('language').
            annotate(lang_count=Count('id'))
        self.modell_count = comments_list.filter(issue_type__id=1).
            count()
        self.total_count = comments_list.count()
        ...

    def get_context_data(self, **kwargs):
        context = super(CommentsList, self).get_context_data(**kwargs)
        context['languages_summary'] = self.languages_summary
        context['modell_count'] = self.modell_count
        context['total_count'] = self.total_count
        return context
```

### Filter by language and issue type

The previously calculated summaries enable filter the comment list by language or by issue type.

*CommentList* view could be called using three different formats of url. For example:

- campaigns/123: Will show all the comments of the campaign 123.
- campaigns/123/lang/ca: Will show comments in catalan language of the campaign 123
- campaigns/123/issue/1: Will show comments with issue type 1 of the campaign 123

These formats are defined in the *url.py* configuration file:

```
path('campaigns/<int:cod_campania_lime>',
     CommentsList.as_view(),
     name='comments_list'),

path('campaigns/<int:cod_campania_lime>/lang/<str:language>',
     CommentsList.as_view(),
     name='comments_list_language'),

path('campaigns/<int:cod_campania_lime>/issue/<int:issue_type>',
     CommentsList.as_view(),
     name='comments_list_issue_type'),
```

The *CommentList*, defined in the *view.py* configuration file, obtains the *cod\_campania\_lime*, *language* and *issue\_type* as parameters and it applies the corresponding filter.

```
class CommentsList(ListView):
    ...

    def get_queryset(self):
        ...
        comments_list = Comment.objects.filter(
            survey__campaign__cod_campania_lime=
            self.kwargs['cod_campania_lime'])
        ...
        if "language" in self.kwargs:
            comments_list = comments_list.filter(
                language=self.kwargs['language'])

        if "issue_type" in self.kwargs:
            comments_list = comments_list.filter(
                issue_type__id=self.kwargs['issue_type'])
        ...
        return comments_list
```

This urls are used in the *comments\_list.html* template:

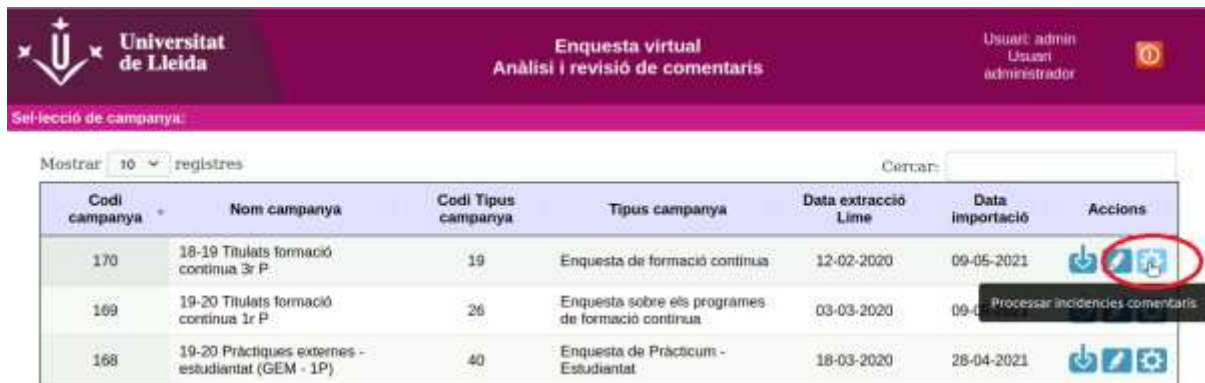
```
...
{% for language_summary in languages_summary %}
    <a href="{% url 'tfmsurveysapp:comments_list_language'
        comments_list.first.survey.campaign.cod_campania_lime
        language_summary.language %}">
        {{ language_summary.lang_count }}
    </a>
    <br/>
{% endfor %}
<a href="{% url 'tfmsurveysapp:comments_list'
    comments_list.first.survey.campaign.cod_campania_lime %}">
    {{ total_count }}
</a>
...
<a href="{% url 'tfmsurveysapp:comments_list_issue_type'
    comments_list.first.survey.campaign.cod_campania_lime 1 %}">
    {{ model1_count }}
</a>
...
```

## Results / Conclusions

As result of the development of this iteration the user interface of the Django tfmsurveys app includes the functionalities developed with Spacy, that enables detect the language of the comments and classify them by issue types.

The user interface had the following visual changes:

The select campaign page has a new button 'Processar comentaris' that executes the process of classifying comments by issue type, or in this case, detect the comments of issue type 1.

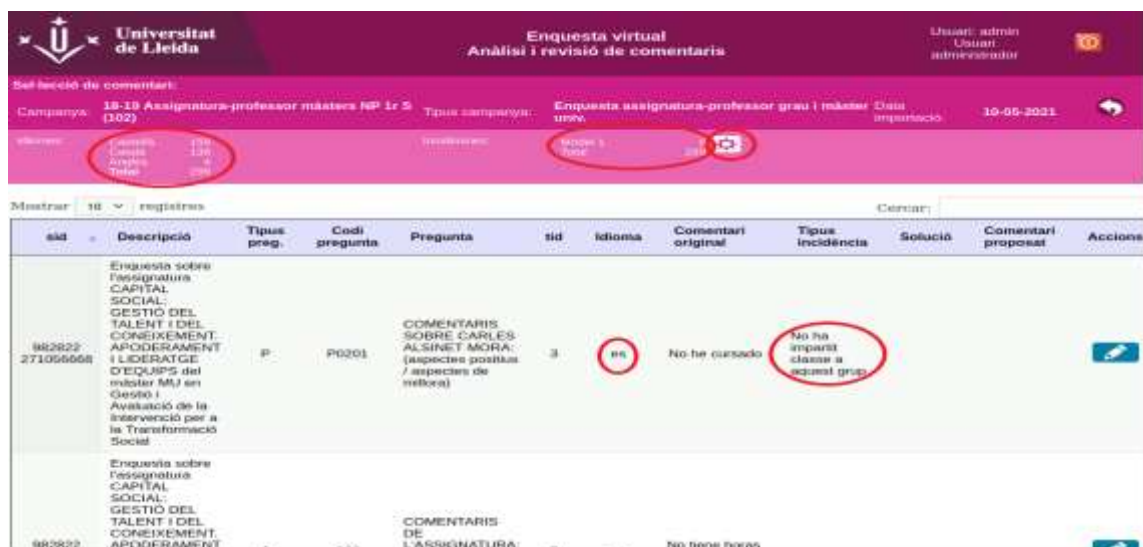


Codi campanya	Nom campanya	Codi Tipus campanya	Tipus campanya	Data extracció Lím	Data importació	Accions
170	18-19 Títolats formació continua 3r P.	19	Enquesta de formació continua	12-02-2020	09-05-2021	[Download] [Edit] [Processar incidències comentaris]
169	19-20 Títolats formació continua 1r P.	26	Enquesta sobre els programes de formació continua	03-03-2020	09-05-2021	[Download] [Edit] [Processar incidències comentaris]
168	19-20 Pràctiques externes - estudiantat (GEM - 1P)	40	Enquesta de Pràcticum - Estudiantat	18-03-2020	28-04-2021	[Download] [Edit] [Processar incidències comentaris]

Image 14 List of campaigns screen modification

The select comment page has some new elements:

- ❖ Summary of comments by language. Clicking in the number the page filter by the selected language
- ❖ Summary of comments by issue type. Only 'model 1' is developed at this moment. Also clicking in the number will filter comments of the issue type 1.
- ❖ Button to process comments and classify them by issue type.
- ❖ New column with the language detected
- ❖ New values in column 'Tipus incidència'



id	Descripció	Tipus preg.	Codi pregunta	Pregunta	id	Idioma	Comentari original	Tipus incidència	Solució	Comentari proposat	Accions
982622-271056068	Enquesta sobre l'assignatura CAPITAL SOCIAL: GESTIO DEL TALENT I DEL CONEIXEMENT. APODERAMENT I LIDERATGE D'EQUIPS del mòdul (M1) en Gestió i Avaluació de la intervenció per a la Transformació Social	P	P0201	COMENTARIS SOBRE CARLES ALSINET MORA: (aspectes positius / aspectes de millora)	3	es	No he cursado	No ha impartit classe a aquest grup			[Edit]
982622	Enquesta sobre l'assignatura CAPITAL SOCIAL: GESTIO DEL TALENT I DEL CONEIXEMENT. APODERAMENT	P	P0201	COMENTARIS DE L'ASSIGNATURA	3	es	No tiene horas				[Edit]

Image 15 Lista of surveys and comments screen modification

The edit comment page has added the possibility of edit the language and the issue type detected:

Universitat de Lleida

Enquesta virtual  
Anàlisi i revisió de comentaris

Usuari: admin  
Usuari administrador

Campanya: 102 18-19 Assignatura-professor màsters NP 1r S Tipus Enquesta assignatura-professor grau i màster univ. Data exportació: 10-04-2019

sid: 982822 Enquesta sobre l'assignatura CAPITAL SOCIAL: GESTIÓ DEL TALENT I DEL CONEIXEMENT, APODERAMENT I LIDERATGE D'EQUIPS del màster MU en Gestió i Avaluació de la Intervenció per a la Transformació Social

Tipus pregunta: P Codi pregunta: P0201 Tid: 3

Pregunta: COMENTARIS SOBRE CARLES ALSINET MORA: (aspectes positius / aspectes de millora)

Comentari original: No he cursado

Idioma: Castellà

Tipus incidència: No ha impartit classe a aquest grup

Proposta solució: -----

Comentari proposat:

Desar

Image 16 Edit comment screen modification

In this document a will create the new chapters of the project documentation and when they are finished a will copy them to the complete document.

## Iteration 9. Model 2. Problematic comments issue

### Requirements / Specification

This implementation consists of the creation of a new Spacy model capable of detecting issues of type 'Problematic comments. This type of comment contains explanations about a problem with a professor or a subject expressed in a correct form. For this reason, it isn't necessary to drop or correct them, but they have to be identified with the aim of reporting them to the direction.

Some of the characteristics of these comments are:

- They are long comments. Contains more than one sentence. In many cases, it contains 3 or 4 sentences.
- They don't follow a clear pattern and they don't have regular expressions clearly identifiable.

Due to these reasons, the comments will be split into sentences with the idea of testing if the Natural Language Processing (NLP) engine will have a better performance with individual sentences than with complete comments. Both approaches will be developed, analyzed and compared.

Another important decision is to choose the size of the data train set and test set. The model generated would be tested with different sizes of sets to compare its performance.

The last variable that can affect results it's the standard model or pipeline chosen. Spacy has usually 3 predefined models for every language. They differ in size and functionalities. We will also compare the performance of the different models for the same language.

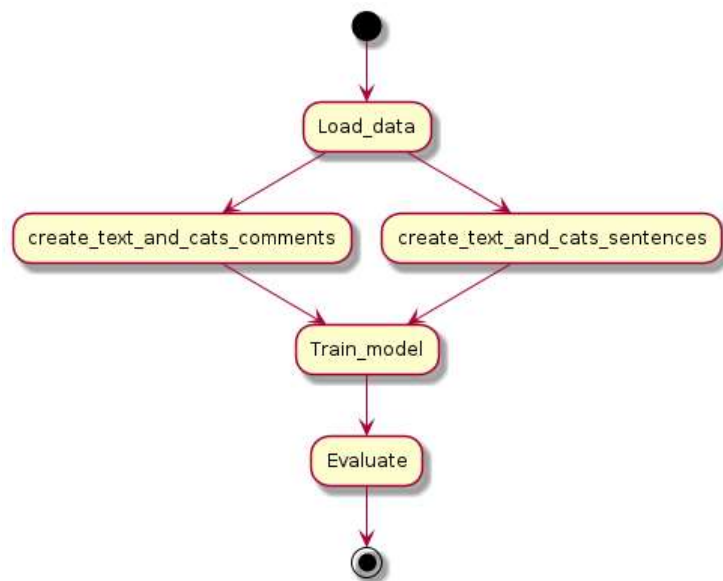
Only the Spanish and Catalan languages will be trained because there are no comments classified in this type of issue in the English language in the original information, which are required to trigger the training.

### Design

The design of this iteration is based on the model 1 iteration design, but adding 2 new functions, ***create\_text\_and\_cats\_comments*** and ***create\_text\_and\_cats\_sentences***.

These functions must split the information from the original files in two lists: *texts* and *cats*. The list *texts* will contain the text of the comment and the list *cats* will contain the label indicating if the comment pertains to the desired type or not.

One of the functions will use the whole comments and the other function will split the comment in the individual sentences. The *load\_data* function will use one or another function depending on the desired approach.



*Schema 27 Activity graph Model 2 processing*

The model will be evaluated with different set sizes, depending on the language and strategy of analysis done. These different sizes will be tested only in the Catalan language.

Language	Strategy	Train + test data size							
Catalan	Comments	6.903	5.000	4.000	3.000	2.000	1.000	700	574
Catalan	Sentences	16.153	10.000	8.000	5.000	4.000	3.000	2.000	1.640

One of the set sizes will be chosen as the main option and tested with 2 standard Spacy models, the smallest and the largest models.

Language	Small model	Large model
Catalan	ca_fasttext_wiki	ca_fasttext_wiki_lg
Spanish	es_core_news_sm	es_core_news_lg

The results of the previous analysis will be used to define the parameters of the Spacy NLP model that must classify the comments of this issue type. In other words will be choosed the approach (full comments or individual sentences), the set size and the base standard model (small or large). This model will be stored in the folder */data/models/model2*.

The analysis of the different options to train the model will be done in the Catalan language



because he has the most comments available. The conclusions of the Catalan language analysis will be applied to the Spanish language, in proportion to his number of comments. The number of comments and sentences in Spanish are:

Spanish	Comments	1.767
Spanish	Sentences	3.809

## Implementation

The implementation of this iteration it's done in the Jupyter Notebook '*model\_2\_comentari\_problematic.ipynb*'. The code it's based on the implementation of the model 1 issue type, done in '*model\_1\_no\_classes.ipynb*'.

The main change it's the creation of the functions '*create\_texts\_and\_cats\_comments*' and '*create\_texts\_and\_cats\_sentences*' and the modification of the '*load\_data*' function to call these functions.

The function '***create\_text\_and\_cats\_comments***' converts the column 'tuples' of the original DataFrame into a list, it shuffles comments and splits them into 2 lists: texts and cats. The code of this function was originally in the *load\_data* function.

An example of tuple is: (*"El professor no s'ajusta als horaris assignats."*, {'cats': {'POSITIVE': True, 'NEGATIVE': False}})

The code of this function is:

```
def create_texts_and_cats_comments(data_com, nlp):

    # Converts data frame to list
    data_com = data_com[data_com["Comentari"].notnull()]
    data_list = data_com["tuples"].tolist()

    # Change order of comments
    random.shuffle(data_list)

    # Split text and label of true cases into two lists
    texts, cats = zip(*data_list)

    return (texts, cats)
```

The '***create\_texts\_and\_cats\_sentences***' function also converts tuples into a list, but loops through the comments of the list and decomposes them into sentences. This action is done by converting the text into a Spacy *doc* object and using the component *sentencizer* of Spacy to split sentences and query them in the *doc.sents* property.

The resulting list is also shuffled and splited into *texts* and *cats* lists, as in the previous function.

This is the code of this function:

```
def create_texts_and_cats_sentences(data_com, nlp):

    # Split comments in sentences
    data_com = data_com[data_com["Comentari"].notnull()]
    train_data = data_com["tuples"].tolist()
    train_list = []
    for row in train_data:
        comment = row[0]
        label = row[1]
        doc = nlp(comment)
        for sent in doc.sents:
            train_list.append((sent.text, label))
        ...

    # Change order of comments
    random.shuffle(train_list)
    ...

    # Split text and label of true cases into two lists
    texts, cats = zip(*train_list)
    ...

    return (texts, cats)
```

The `load_data` function has also changed to call the previous functions and to show more clear information in the log. Some commands had changed their order to allow calling these functions.

This is the main new part of this function:

```
def load_data(limit = 0, split = 0.8, language = "es", nlp = None):
    ...

    (texts_true, cats_true) = create_texts_and_cats_comments(
        data_true, nlp)
    # (texts_true, cats_true) = create_texts_and_cats_sentences(
        data_true, nlp)
    ...
```

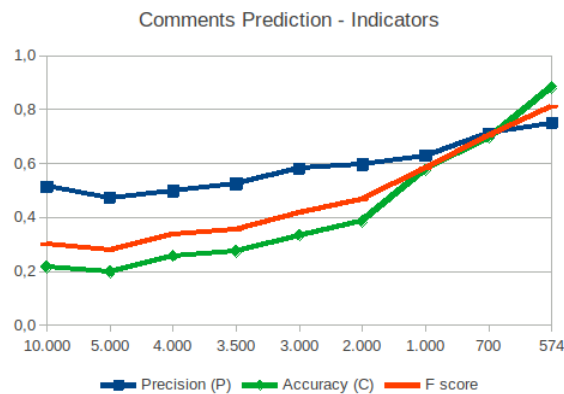
Some little modifications are made in the `train_model` function to clarify the information in the log of the execution.

## Results / Conclusions

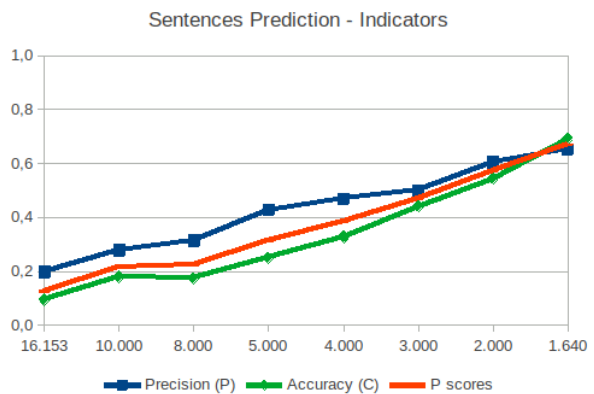
As a result of this iteration, an NLP model has been developed using Spacy that will classify comments of issue type 2 - Problematic Comments.

This model has been evaluated using 2 strategies: full comments or individual sentences. Every strategy has been evaluated in different set sizes.

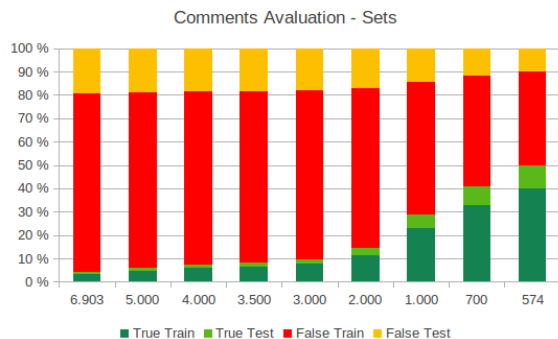
The following graphs show the results of this evaluation and the internal distribution of the sets, in train and test subsets and true or false labelled comments.



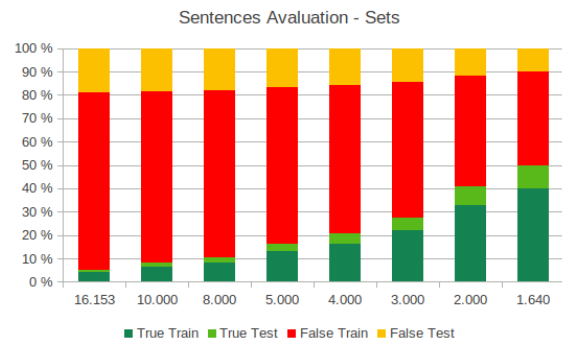
Schema 28 Comment prediction – indicators graph



Schema 29 Sentence prediction – indicators graph



Schema 30 Comments avaluation - sets graph



Schema 31 Sentence avaluation - sets graph

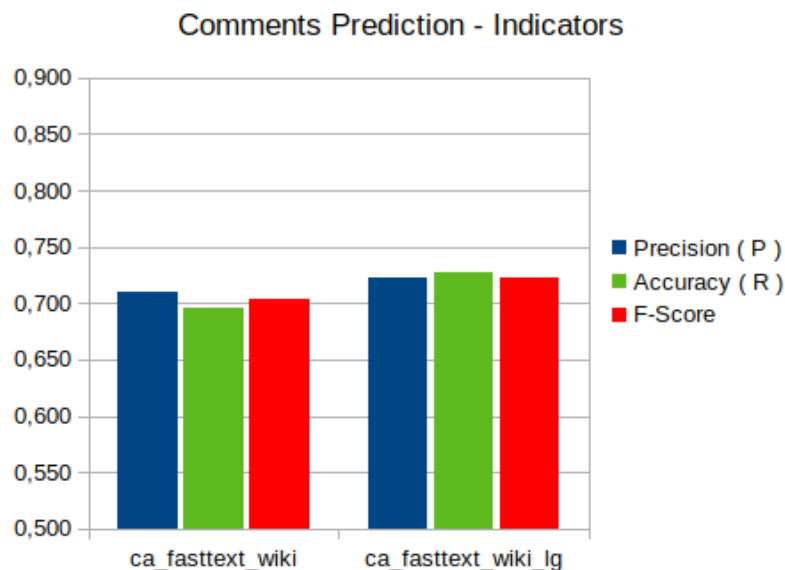
Against expectations, results of full comments analysis are better than individual sentences analysis. Might be this result is related to the characteristics of this type of issue. As explained in the introduction this type of comments usually are long and includes some sentences. Using full comments in the training, the model could detect this characteristic and use it in the prediction.

It could be also seen that all the indicators increase their value when the set size decreases. This could be due to the fact the proportion of true and false comments change. The best results are obtained when the proportion of false comments is lower.

Additionally, with a size of **700 comments**, the precision and accuracy lines cross and the results are equilibrated between them. This is the optimal size, where the final model will be trained. In this set size, the proportion of true comments reaches 40%.

Spacy allows the selection between some pre-trained models or pipelines that contain different amounts of information. Until now the model has been trained using the smallest model (*ca\_fasttext\_wiki*). On this occasion, the model will be trained with the larger model (*ca\_fasttext\_wiki\_lg*), with the purpose of analyzing if there is any difference in the results of the evaluation.

This is the comparison between the two pre-trained models:



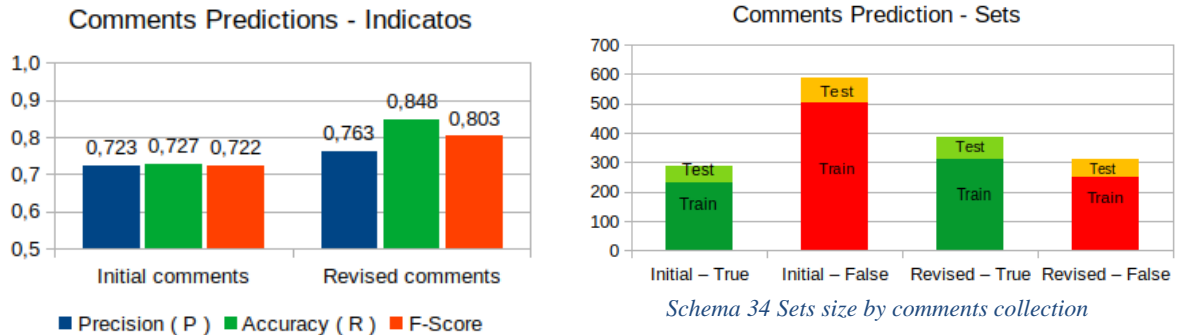
*Schema 32 Pretrained models comparison graph*

This new evaluation has obtained a 2% better performance in the F-score. It's not a big increment but interesting enough to choose the model for the final training.

After executing the model and evaluating the model it's possible to show most of the **False positives** detected could be True positives. Due to the fact that the comments are labelled by a user in a subjective form, this classification could be enhanced.

The classification of the comments in the original data has been revised using the information of the False positives and a significant set of them has been labelled as 'Comentari problemàtic'. With this work, 100 new comments have been obtained.

The model has been trained with 700 comments, *ca\_fasttext\_wiki\_lg* pre-trained model and new true comments. These new comments result in the following results.

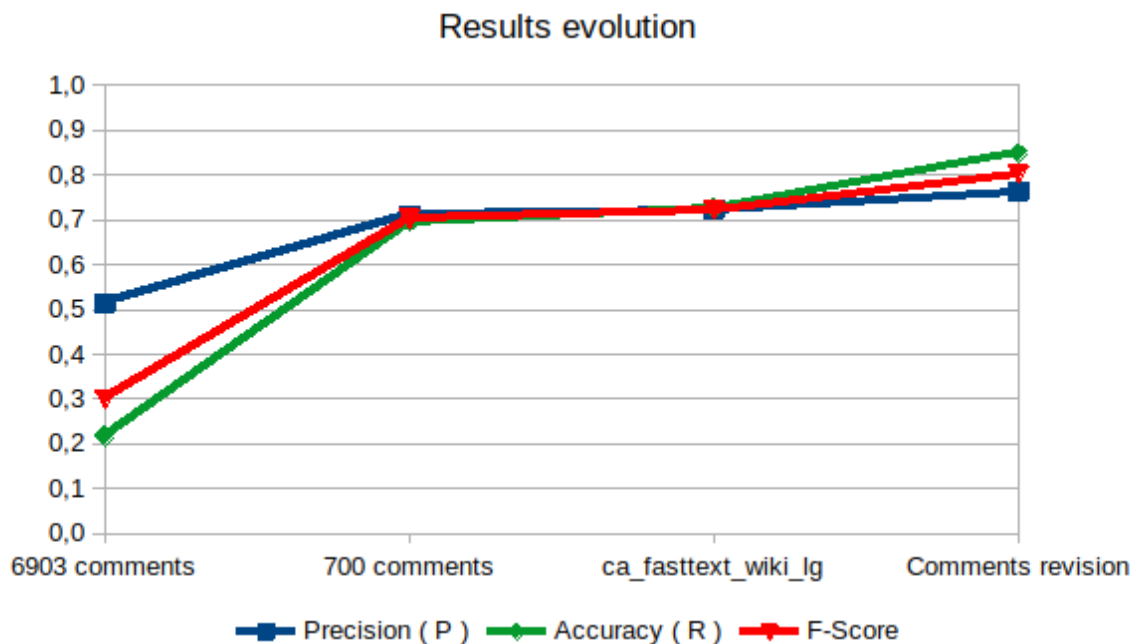


Schema 33 Indicators graph by comments collection

Schema 34 Sets size by comments collection

Results obtained with the new set of comments increase by 8% in F-Score, but they are better in Accuracy, where they increase by 12%. We can also appreciate the variation in the composition of the training test sets.

This final comparative shows how has evolved results with the different solutions applied:



Schema 35 Indicators evolution by solution type

- 6903 comments: Using all the comments in the original data only a 30% of F-score could be achieved.
- 700 comments: Using only a part of the comments, and selecting the set size where the 3 indicators cross, the model achieved an F-Score of 70%
- ca\_fasttext\_wiki\_lg: Using the larger pre-trained model the model increases his F-score to 72%
- Comments revision: After reclassifying most of the False positives the result rises to 80% in F-Score and 85% in Accuracy ( R ).

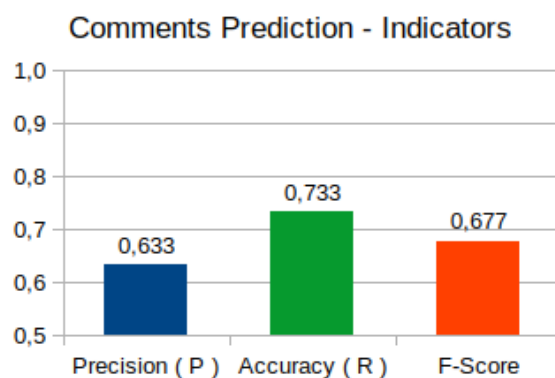
In the case of the **Spanish** language the size of the sets has been calculated in proportion to the Catalan model and they will contain full comments.

The Spanish original comments have only 53 cases labelled as True. These comments will be all used and divided into the train set (80%) and test set (20%). Comments labelled as False must be 60% of the comments, 80 comments, and also must be divided into train and test sets. The False comment will be selected in every loop of the training from the 1.714 total False comments.

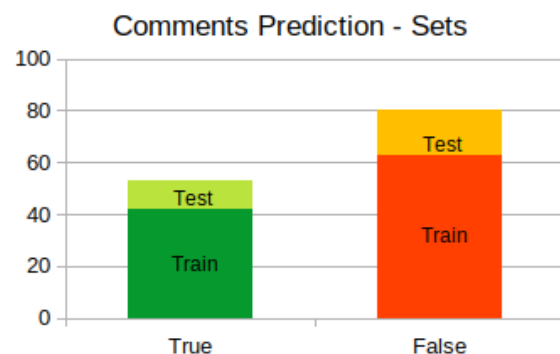
The final distribution is:

<b>Comments</b>	1.767
<b>Sentences</b>	3.809

	<b>Total</b>	<b>Train (80%)</b>	<b>Test (20%)</b>
<b>True comments</b>	53	42	11
<b>False comments</b>	80	63	17
<b>Total</b>	133	106	27



*Schema 36 Indicators graph in Spanish language*



*Schema 37 Set sizes in Spanish language*

The results of the evaluation of this model in the Spanish language are discrete because of the little information available.

## Iteration 10. Integration of Django and Spacy in Model 2 - Problematic comments

### Requirements / Specification

This iteration will integrate the second model created with *Spacy* in the *Django* app. This integration consists of creating a new class that loads the NLP model and test comments to classify them as included in issue type 2, 'Problematic comments'.

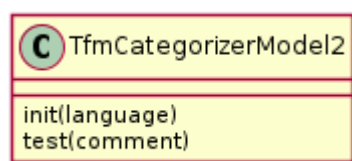
The user interface must also change to execute the processing function and show the results. In the same way, the filter by issue type 2 must be implemented.

Another potential problem we have to consider is the processing time and the feedback with the user. The time needed to process all the comments in a campaign and to classify them in both models could be long. Therefore, we must study executing the classification in an asynchronous task and informing the user about the processing state.

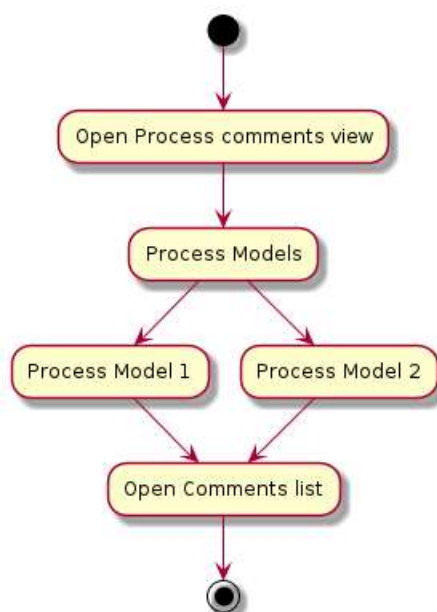
### Design

This new integration will be based on the previous integration between *Spacy* and *Django*.

A new class will be created: ***TfmCategorizerModel2***. It will include two methods:



- **init:** Load the Spacy trained model of the selected language.
- **test:** Receive a comment and calculate the probability to be classified as issue type 2.



The detection of the issue type 2 will be also done in the view *Process comments*.

A new function ***Process Models*** will be created and it will include the previous function Process Model 1 and a new *Process Models 2* function. The function Process Models 2 will be based on the Process Models 1, but classifying comments as issue type 2 - Problematic comments. Only the catalan and spanish languages will be processed.

When the processing ends the *Comments list* view will be opened.

Schema 38 Activity graph Model 2 processing

## Implementation

For the implementation of this iteration has been done the following modifications in the previous code related with processing comment.

But the most significant contribution of this iteration is creating an asynchronous task that executes the comments processing. We will use to new tools:

- Celery 5.1: Queue's task manager
- RabbitMQ 3.8.2: Message transport broker.

### Creation of class TfmCategorizerModel2

The class TfmCategorizerModel2 is based on the class TfmCategorizerModel1 and has also two functions as described in the design:

- **init:** Loads the model created in the training iteration. Every language has its own model placed in a subfolder of the folder *tfmsurveysapp/models*. For example: *nlp\_models/model2/ca* and *nlp\_models/model2/es*. Doesn't exist a model for the English language.
- **test:** This function creates a *Spacy doc* object from the received comment. This object contains the *cats* object that indicates the probability that the comment matches with the issue type '2 - Comentari problemàtic'.

Example: "Sembla que tingui ganes de que patim, i ens tracta de tontos, {'POSITIVE': 0.9999703168869019, 'NEGATIVE': 2.9632559744641185e-05}"

This is the code of the class:

```
class TfmCategorizerModel2():  
  
    def __init__(self, language):  
        ...  
        self.nlp = spacy.load(pathmodel + language)  
        ...  
  
    def test(self, comment):  
        doc = self.nlp(comment)  
        ...  
        return doc.cats
```

### Creation of Celery task process\_models

Processing the comments is hard work that could take a lot of time. If it is executed inside the view the system will be blocked until the processing ends and the user won't have information about the state of the execution. This situation may lead to insecurity in the user and the feeling that something is going wrong.



To avoid this situation, we must execute the processing of comments as an asynchronous task that can provide state information to the user. The best way to use tasks in Django is the tool **Celery**.

Celery is a distributed system to process messages. It provides a task queue focused on real-time processing, but also supporting task scheduling. A dedicated worker processes constantly monitor task queues to execute new work.

Celery requires a message transport to send and receive messages. We will use **RabbitMQ** that is fully integrated with Celery.

We will need some installation and configuration steps.

The first speed is to install RabbitMQ and execute the server:

```
sudo apt-get install rabbitmq-server  
sudo rabbitmq-server
```

This will configure the rabbitmq-server as a daemon that will run with when the system starts.

The following step is installing Celery:

```
pip3 install celery
```

To start the Celery tasks queue, we must execute the following command:

```
celery -A tfmsurveys worker --loglevel=INFO
```

To use Celery with Django we must define it as an App in the file `__init__.py`

```
from .celery import app as celery_app  
  
__all__ = ('celery_app',)
```

Then we must create a **celery.py** module to define the instance of Celery.

```
import os  
  
from celery import Celery  
  
# Set the default Django settings module for the 'celery' program.  
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'tfmsurveys.settings')  
  
app = Celery('tfmsurveys')
```

```
# - namespace='CELERY' means all celery-related configuration keys
#   should have a `CELERY_` prefix.
app.config_from_object('django.conf:settings', namespace='CELERY')

# Load task modules from all registered Django apps.
app.autodiscover_tasks()

@app.task(bind=True)
def debug_task(self):
    print(f'Request: {self.request!r}')
```

And the most important step is creating the module **tasks.py** where we will write all our tasks. In our case it will contain the **process\_models\_task**.

This is the main code of the **tasks.py**, related with the **process\_models** task.

```
from celery import shared_task, Celery
from celery_progress.backend import ProgressRecorder
from tfmsurveysapp.models import IssueType, Comment, Campaign
from tfmsurveysapp.spacy.model_1_execution import TfmCategorizerModel1
from tfmsurveysapp.spacy.model_2_execution import TfmCategorizerModel2

@shared_task(bind=True)
def process_models_task(self, cod_campania_lime):
    ...
    process_model1(cod_campania_lime, campaign)
    process_model2(cod_campania_lime, campaign)
    ...

    return True

def process_model1(cod_campania_lime, campaign):
    ...

def process_model2(cod_campania_lime, campaign):
    ...
```

The **process\_model1** function is the same as in the previous iteration.

The **process\_model2** function is based on the **process\_model2** function, but English language won't be processed. This is the main part of its code:

```
def process_model2(cod_campania_lime, campaign):

    issue_type2 = IssueType.objects.get(id=6)
    languages = {"ca", "es", "en"}
    for language in languages:
        if language != "en":
            ...
            nlp2 = TfmCategorizerModel2(language)

    comments = Comment.objects.filter(
        survey_campaign_cod_campania_lime=cod_campania_lime,
        language=language)
```

```

...
positives2 = 0
for comment in comments:
    if language != "en":
        result2 = nlp2.test(comment.original_value)
        if result2['POSITIVE'] > 0.5:
            positives2 = positives2 + 1;
        ...

return True

```

### Given information to the user

To inform the user of the state of the processing of the comments, I have taken the following strategy:

- Two new fields (*estat* and *subestat*) had been added to the *Campaign* model.
- Inside the *process\_models* task, we add information in these fields, informing of the state of the calculation.
- The fields *estat* and *subestat* are shown in the *comments\_list.html* template. They are also used to show or hidden elements in the template.
- A 'close information' button deletes the remaining information after the process has finished.
- A timer has been added to the *comments\_list.html* template to refresh the information every 5 seconds.

These are the changes in *models.py*:

```

class Campaign(models.Model):
    ...
    estat = models.CharField("Estat", max_length=50, null=True)
    subestat = models.CharField("Subestat", max_length=50, null=True)
    ...

```

This is the information added in *tasks.py*:

```

def process_models_task(self, cod_campania_lime):

    campaign = Campaign.objects.get(cod_campania_lime=cod_campania_lime)
    campaign.estat = "Processant comentaris..."
    campaign.subestat = "Iniciant process de calcul"
    campaign.save()

    process_model1(cod_campania_lime, campaign)
    process_model2(cod_campania_lime, campaign)

    campaign.estat = "Comentaris processats"
    campaign.subestat = ""
    campaign.save()

    return True

```

```

def process_model1(cod_campania_lime, campaign):

    issue_type1 = IssueType.objects.get(id=1)
    languages = {"ca", "es", "en"}
    for language in languages:
        print ("ProcessComments: Reading model: ", language )
        campaign.subestat = "Llegint model 1: " + language
        campaign.save()
        nlp1 = TfmCategorizerModel1(language)

        comments =
Comment.objects.filter(survey__campaign__cod_campania_lime=cod_campania_lime, language=language)
        print("ProcessComments: Processing comments", language)
        campaign.subestat = "Processant comentaris model 1: " + language
        campaign.save()

        positives1 = 0
        for comment in comments:
            ...

    return True

```

These are the changes in ***comments\_list.html***:

```

{% if comments_list.first.survey.campaign.estat > "" %}
    <table width="100%" border="0" cellspacing="0" cellpadding="0"
        class="Info">
        <tr valign="top">
            <td width="6%">&nbsp;  </td>
            <td align="center" >
                <b>{{ comments_list.first.survey.campaign.estat }}</b>
                &nbsp;  
                {{ comments_list.first.survey.campaign.subestat }}
                &nbsp;  
            </td>
            <td width="6%" align="center">
                {% if comments_list.first.survey.campaign.estat ==
                    "Comentaris processats" %}
                <a href="{% url 'tfmsurveysapp:close_info'
                    comments_list.first.survey.campaign.
                    cod_campania_lime %}" target="_self"
                    onmouseover="MM_swapImage('btnCerrarInfo','',
                    '{% static 'img/boton_cerrar_info_up_32.png' %}',
                    1)"
                    onmouseout="MM_swapImgRestore()"
                    style="cursor: pointer">
                    
                </a>
                {% endif %}
            </td>

```

```
</tr>
</table>
```

This is the code related with the **timer** in *comments\_list.html*:

```
script language="JavaScript" type="text/javascript" class="init">
$(document).ready(function() {

{% if comments_list.first.survey.campaign.estat ==
    "Processant comentaris..." %}
    setTimeout(refrescar, 5000);
{% endif %}

function refrescar() {
    {% if comments_list.first.survey.campaign.estat ==
        "Processant comentaris..." %}
        url = "{% url 'tfmsurveysapp:comments_list'
            comments_list.first.survey.campaign.cod_campania_lime %}"
        window.location.href = url;
    {% endif %}
}

</script>
```

## Changes in summary of issue types

The view **CommentList** has been modified to calculate the number of comments detected as issue type 2 and pass this value to the template, using the context,

These are changes in view:

```
class CommentsList(ListView):
    ...
    def get_queryset(self):
        ...
        self.model2_count = comments_list.filter(
            issue_type__id=2).count()
        ...
        return comments_list

    def get_context_data(self, **kwargs):
        ...
        context['model2_count'] = self.model2_count
        ...
        return context
```

The template *comments\_list.html* has been modified to show the number of comments of issue type 2 and filter them.

These are changes in template:

```
<tr>
  <td class="EtiquetaSummary" nowrap="nowrap">
    Model 2 - Comentari problem&agrave;tic
  </td>
  <td align="center" class="EtiquetaSummary">
    <a href="{% url 'tfmsurveysapp:comments_list_issue_type'
      comments_list.first.survey.campaign.cod_campania_lime 2 %}">
      {{ model2_count }}
    </a>
  </td>
</tr>
```

## Results / Conclusions

After finishing this iteration we have completed the implementation of the app Tfmsurveys.

In this iteration, we have added in the processing of comments the classification in the issue type 2 - Problematic comment, using the NLP Spacy model created previously.

We have also implemented the use of tasks to process the comment. This implementation has been done with the tools Celery and RabbitMQ.

In addition, we have developed a method to give information to the user about the state of the process.

When the user starts the comments processing, the information panel appears with the values of the state and substate saved in the campaign.

The user can work with the application while the server is processing.

Every campaign has independent states, and the user can launch more than one task of processing comments, but the application's performance will go down.



Universitat de Lleida

Enquesta virtual

Anàlisi i revisió de comentaris

Usuari: admin

Usuari administrador



Sel·lecció de comentari:

Campanya:

18-19 Pràctiques externes - estudiantat (GEM - 3P) (132)

Tipus campanya:

Enquesta de Pràcticum - Estudiantat

Data importació:

15-06-2021



Idiomes:

Català

25

Total

25

Incidències:

Model 1 - No classes

0

Model 2 - Comentari problemàtic

2

Total

25



Processant comentaris... iniciant process de càlcul

Mostrar 10 registres

Cercar:

sid	Descripció	Tipus preg.	Codi pregunta	Pregunta	tid	Idioma	Comentari original	Tipus incidència	Solució	Comentar proposat
844391 271056704	Enquesta de satisfacció sobre PRÀCTIQUES EXTERNES EN ENTITATS PÚBLIQUES I PRIVADES del 'Doble' titulació: Grau en Enginyeria Informàtica i		P013	<div> <div>&lt;p&gt; &lt;span style="font-family:arial,helvetica,sans-serif;"&gt;&lt;em&gt;Podeu introduir els vostres comentaris sobre el programa de pràctiques: &lt;/em&gt;&lt;/span&gt;&lt;/p&gt;</div> </div>	4	ca	<div>Àrees de millora: Crec que, en l'avaluació de les pràctiques, hauria de tenir més pes la part avaluativa del tutor de l'entitat doncs és qui fa el seguiment real de l'activitat de l'alumne.</div>			

Image 17 Information about processing start

The step that takes more time in the processing is the loading of the models.

**Universitat de Lleida** Enquesta virtual Anàlisi i revisió de comentaris

Usuari: admin Usuari administrador

**Selecció de comentari:**

Campanya: 18-19 Pràctiques externes - estudiantat (GEM - 3P) (132) Tipus campanya: Enquesta de Pràcticum - Estudiantat Data importació: 15-06-2021

Idiomes: Català: 25 Total: 25 Incidents: Model 1 - No classes: 0 Model 2 - Comentari problemàtic: 1 Total: 1

Processant comentaris... Llegint model 1: ca

sid	Descripció	Tipus preg.	Codi pregunta	Pregunta	tid	Idioma	Comentari original	Tipus incidència	Solució	Comentari proposat
844391 271056704	Enquesta de satisfacció sobre 'PRÀCTIQUES EXTERNES EN ENTITATS PÚBLIQUES I PRIVADES' del 'Doble titulació: Grau en Enginyeria Informàtica i		P013	<p> <span style="font-family: arial, helvetica, sans-serif;">Podeu introduir els vostres comentaris sobre el programa de pràctiques: </em></span></p>	4	ca	Àrees de millora: Crec que, en l'avaluació de les pràctiques, hauria de tenir més pes la part avaluativa del tutor de l'entitat doncs és qui fa el seguiment real de l'activitat de l'alumne.			

Image 18 Information about processing model 1

The task reads 5 models: Model 1 - catalan, Model 1 - spanish, Model 1 - english, Model 2 - catalan and Model 2 - spanish.

The order of the languages can change in every execution.

The screen refreshes every 5 seconds.

**Universitat de Lleida** Enquesta virtual Anàlisi i revisió de comentaris

Usuari: admin Usuari administrador

**Selecció de comentari:**

Campanya: 18-19 Pràctiques externes - estudiantat (GEM - 3P) (132) Tipus campanya: Enquesta de Pràcticum - Estudiantat Data importació: 15-06-2021

Idiomes: Català: 25 Total: 25 Incidents: Model 1 - No classes: 1 Model 2 - Comentari problemàtic: 1 Total: 2

Processant comentaris... Llegint model 2: ca

sid	Descripció	Tipus preg.	Codi pregunta	Pregunta	tid	Idioma	Comentari original	Tipus incidència	Solució	Comentari proposat
844391 271056704	Enquesta de satisfacció sobre 'PRÀCTIQUES EXTERNES EN ENTITATS PÚBLIQUES I PRIVADES' del 'Doble titulació: Grau en Enginyeria Informàtica i		P013	<p> <span style="font-family: arial, helvetica, sans-serif;">Podeu introduir els vostres comentaris sobre el programa de pràctiques: </em></span></p>	4	ca	Àrees de millora: Crec que, en l'avaluació de les pràctiques, hauria de tenir més pes la part avaluativa del tutor de l'entitat doncs és qui fa el seguiment real de l'activitat de l'alumne.			

Image 19 Information about processing model 2

When the processing of comments finishes then the app informs the users and the close comments button appears. This button removes the information of the state and substate field and hides the information panel.

The screenshot shows the 'Enquesta virtual' interface with the title 'Anàlisi i revisió de comentaris'. The user is logged in as 'admin administrador'. The selected campaign is '18-19 Pràctiques externes - estudiantat (GEM -3P) (132)'. The campaign type is 'Enquesta de Pràcticum - Estudiantat'. The data import date is '15-06-2021'. The interface shows a summary of comments: 25 total, 25 processed, 0 incidents, and 2 problematic comments. A yellow bar indicates 'Comentaris processats'. Below this, a table displays the processed comments.

sid	Descripció	Tipus preg.	Codi pregunta	Pregunta	tít	Idioma	Comentari original	Tipus incidència	Solució	Comentari proposat	Acció
844391 271056702	Enquesta de satisfacció sobre PRÀCTIQUES EXTERNES EN ENTITATS PÚBLIQUES I PRIVADES del Grau en Enginyeria Informàtica i Grau en ADE i el tutoria.		P013	<p><span style="font-family:arial,helvetica,sans-serif;"><em>Podeu introduir els vostres comentaris sobre el programa de pràctiques:</em></span></p>	2	ca	En el cas de cursar les pràctiques a una empresa tecnològica, crec que el millor seria assignar un tutor amb algun coneixement relacionat en el sector.	Comentari problemàtic			

Image 20 Information process finished



# Conclusions

Along this project, an application has been developed that can help the Quality and Teaching Planning unit of the UdL manage the comments written by the students in the teaching surveys.

This tool is a web application developed in Python using the framework Django. This application uses Natural Language Processing (NLP), a subfield of Artificial Intelligence (AI), to process comments. Spacy is the NLP library selected to implement this task.

The requirements definition and the development of the application follow the Agile Behaviour Driven methodology.

The final web app has the **features**:

- Importation of survey campaign from the original application Lime Survey.
- List and search the available campaigns.
- Importation of surveys and comments of a campaign from Lime Surveys.
- List and search the surveys and comments of a campaign
- Processing of the comments to identify their language and classify them in the issue types 'Professor taught this group' and 'Problematic comments'
- Manual edition of comments.

The use of **Agile Behaviour Driven Development** methodology in the requirements definition helped communicate with the final users because it uses a language structured but easy to understand to non-computing experts.

We must remember this methodology uses the word 'features' to define the high level of abstraction and 'scenarios' to describe the detailed situations.

It's practical to define features, forcing the user to answer the questions: In order to?, As a? and I want to? And the questions to define the scenarios: Given? When? Then?

The implementation of the user interface was done using the **Django** framework. The main characteristic of this framework is that it divides the development in the following layers:

- Models: they include the definition of the objects and properties corresponding to the tables and fields of the database. Models automate the management of information between the application and the database.
- URLs and Views: they control the flux of information between the pages of the application and execute the business logic.
- Templates (HTML): they define the user interface format and collect information from views.

This is a superficial description of a complex structure that includes a lot of functionalities to help in the development of applications.

I have explored some of them:

- Legacy database access
- Multiple database management
- Multifield keys issue
- Virtual model
- Use of RedirectView
- Batch of bulk operations
- Raw queries

It was interesting to know the possibilities of this powerful framework.

The processing of the comments was done using the NLP library **Spacy**. This library includes a lot of functionalities in the analysis and manipulation of text in natural language. I tried a part of them in the development of the features:

- Language recognition. The original information doesn't contain the language of the comments, and it must be determined. I use the class `LanguageDetector` of Spacy and the library `PyCld2` to detect the language. But I need to combine both to improve their individual results.
- Classification of comments of the issue type 'Professor didn't teach this group'. I explored two options to classify comments:
  - Use of the statistical model with the class `TextCategorizer`.
  - Analysis of patterns with Part-of-Speech and Dependency Parsing

After comparing the results, I chose `TextCategorizer`. Part-of-Speech has many possibilities but requires much more work of customizing patterns.

- Classification of comments of the issue type 'Problematic comments'. In that case, I use only `TextCategorizer`, but with the following considerations:
  - Classification of full comments or individual sentences.
  - Size the training and test sets.
  - Use of small or large pre-trained models available in each language.

The selected option was to use full comments, a relatively small set and large models.

One of the problems in the implementation of the models was that insufficient sample data was available.

In my opinion it was very interesting work with the Spacy library. I found it easy to use and with a lot of functionalities.

The last tools used in this project are **Celery** and **RabbitMQ**. They are a queue's task manager and a message transport broker. I used them to execute the processing of comments in an asynchronous way and to keep the user informed about the state of the process.

As regards the achievement of the **objectives**, they slightly changed during the project. In general, I have discarded some functionalities of the application to increase the exploration of

new technologies.

The functional objectives excluded are:

- Proposal of alternatives or removal of comments.
- Detection of the most used expressions (trending topics)
- Graphical analysis of the results with queries modifiable by the user.
- Detection of only two issue types.

On the other hand, I have expanded my work in:

- Learn a lot of the instruments provided by Django in the programming of applications.
- Try and analyze different methods to classify texts and analyze their grammar using Spacy.

This change in the objective is also reflected in the **scheduling** of the project.

Each iteration took more or less one month, and I added new iterations to search and analyze alternatives.

Every iteration was different from the previous one and it supposed to be a new challenge for me. I need to learn new techniques and look for innovative solutions all the time.

It was a great experience to learn about all these tools and use them to resolve a real problem.

## Future development

This project has a lot of possibilities to grow in the future.

The first need is to increase the set of sample comments. We could add more comments from the campaigns 2020-21 and ask users to label them. This action will significantly enhance the accuracy of predictions.

Other possible enhancements are the functionalities not included in the project:

NLP model creation for the other type of issues:

- Offensive comments.
- Teacher comment on subject question
- Subject comment in teacher's question
- Spelling mistakes
- Exclamation points and excessive emoticons

Execute different types of action to solve the issues detected. It could consist in deleting comments, proposing alternatives, moving comments from a survey to another, correct spelling, etc.

Next step would be testing the application with the final users and maybe use it in a production environment. In this process users could be asked to make suggestions and proposals to improve the application.

Technical users and directive staff of the UdL want to know the results of the comments detection. It could be implemented reports that show comments classified for issue type and grouped or filtered by center, degree, professor or subject, for example.

There are a lot of comments without issues related to each degree or center, and it's a hard work reading them and extracting a conclusion about his global sense. It could be a great help to detect the trending topics or obtain an automatic summary of a set of comments from a campaign.

Next step would be testing the application with the final users and maybe use it in a production environment. In this process users could be asked to make suggestions and proposals to improve the application.

# Bibliography

<https://www.agilealliance.org/glossary/bdd/>, What is BDD (Behavior Driven Development)?, Agile Alliance

<https://dannorth.net/introducing-bdd/>, Introducing BDD, Dan North & Associates

<https://plantuml.com>, herramienta de código abierto - PlantUML, PlantUML

<https://docs.djangoproject.com/en/3.1/>, Django documentation, Django Software Foundation

<https://www.youtube.com/playlist?list=PLJ0YJaEOtqlkdqTHfeYT4kVmcA4p8dfIr>, Django Web Project Tutorial, Roberto Garcia

<https://developer.mozilla.org/es/docs/Learn/Server-side/Django>, Framework Web Django (Python) - Aprende sobre desarrollo web | MDN, Mozilla

<https://data-flair.training/blogs/django-redirect/>, Django Redirects – The Essential Guide of URL Redirects in Django, DataFlair

<https://www.protechtraining.com/blog/post/tutorial-using-djangos-multiple-database-support-477>, Tutorial: Using Django's Multiple Database Support, ProTech Professional Technical Services, Inc.

<https://www.limesurvey.org/es/soluciones/universidades>, Universidades - LimeSurvey - Easy online survey tool, LimeSurvey GmbH

<https://spacy.io>, Industrial-Strength Natural Language Processing, ExplosionAI GmbH

<https://medium.com/towards-artificial-intelligence/natural-language-processing-with-spacy-steps-and-examples-155618e84103>, Natural Language Processing with Spacy, Dhilip Subramanian, Medium.com

<https://www.machinelearningplus.com/nlp/custom-text-classification-spacy/>, How to Train Text Classification Model in spaCy?, Machine Learning Plus

<https://github.com/ccoreilly/spacy-catala>, Spacy NLP Model for the Catalan language, Ciaran O'Reilly

<https://pypi.org/project/pyclld2/>, PYCLD2 - Python Bindings to CLD2, Python Software Foundation

<https://www.nltk.org>, Natural Language Toolkit — NLTK 3.6.2 documentation, NLTK Project

<https://pub.towardsai.net/text-mining-in-python-steps-and-examples-78b3f8fd913b>, Text mining in Python: Steps and examples: NLTK, Dhilip Subramanian, Medium.com

<https://stanfordnlp.github.io/CoreNLP/>, Overview - CoreNLP, Stanford NLP group

<https://allennlp.org>, AllenNLP

<https://www.ibm.com/cloud/learn/natural-language-processing>, What is Natural Language Processing, IBM

<https://docs.celeryproject.org/en/latest/getting-started/index.html>, Getting Started — Celery 5.1.0 documentation, Ask Solem

<https://docs.celeryproject.org/en/latest/django/first-steps-with-django.html>, First steps with Django — Celery 5.1.0 documentation, Ask Solem

<https://docs.celeryproject.org/en/master/getting-started/backends-and-brokers/rabbitmq.html>, Using RabbitMQ — Celery 5.1.0 documentation, Ask Solem

<https://www.rabbitmq.com/getstarted.html>, RabbitMQ Tutorials — RabbitMQ, VMware, Inc.

<https://github.com/czue/celery-progress>, czue/celery-progress: Drop in, configurable ... - GitHub, Cory Zue (czue)